

Run-Time Library Reference

© 1999 Sony Computer Entertainment Inc.

Publication date: September 1999

Sony Computer Entertainment America
919 E. Hillsdale Blvd., 2nd floor
Foster City, CA 94404

Sony Computer Entertainment Europe
Waverley House
7-12 Noel Street
London W1V 4HH, England

The *Run-Time Library Reference* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® License and Development Tools Agreements, the Licensed Publisher Agreement and/or the Licensed Developer Agreement.

The *Run-Time Library Reference* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® License and Development Tools Agreements, the Licensed Publisher Agreement and/or the Licensed Developer Agreement.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® License and Development Tools Agreements, the Licensed Publisher Agreement and/or the Licensed Developer Agreement.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *Run-Time Library Reference* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

PlayStation and PlayStation logos are registered trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

CONFIDENTIAL

Summary Table of Contents

About This Manual

Changes Since Last Release	v
Related Documentation	vi
Manual Structure	vi
Developer Reference Series	vii
Typographic Conventions	viii
Developer Support	viii

Chapter 1: Kernel Library

Structures	1-3
Functions	1-9

Chapter 2: Standard C Library

Functions	2-3
-----------	-----

Chapter 3: Math Library

Functions	3-3
-----------	-----

Chapter 4: Memory Card Library

Functions	4-3
-----------	-----

Chapter 5: Extended Memory Card Library

Functions	5-3
-----------	-----

Chapter 6: Data Compression Library

Structures	6-3
Functions	6-5

Chapter 7: Basic Graphics Library

Structures	7-5
Functions	7-33
Macros	7-118

Chapter 8: Basic Geometry Library

Structures	8-5
Functions	8-21

Chapter 9: Extended Graphics Library

Structures	9-3
Functions	9-31
Macros	9-109
External Variables	9-112

Chapter 10: CD/Streaming Library

Structures	10-3
Functions	10-8

Chapter 11: Extended CD-ROM Library

Structures	11-3
Functions	11-7

Chapter 12: Controller/Peripherals Library

Functions	12-3
-----------	------

Chapter 13: Link Cable Library

Functions	13-3
Macros	13-8

Chapter 14: Extended Sound Library	
Structures	14-5
Functions	14-14
Chapter 15: Basic Sound Library	
Structures	15-5
Functions	15-15
Chapter 16: Serial Input/Output Library	
Functions	16-3
Chapter 17: HMD Library	
Structures	17-3
Functions	17-23
Chapter 18: PDA Library (libmcx)	
Functions	18-3
Chapter 19: Memory Card GUI Module (mcgui)	
Structures	19-3
Functions	19-10
Index	

About This Manual

This manual is the latest release of the PlayStation® *Library Reference* as of Run-Time Library release 4.6. The purpose of this manual is to define all available PlayStation run-time library functions, macros and structures. The companion *Run-Time Library Overview* volume describes the structure and purpose of the libraries in programming software for the PlayStation.

Changes Since Last Release

This manual has been expanded in content since release 4.5 of the Run-time Library. It combines all previously released material with the latest Run-time Library 4.6 information.

Kernel Library (Chapter 1)

Functions revised:

FlushCache()
StartPAD()

Data Compression Library (Chapter 6)

Function added:

EncSPU2()

Function Revised:

ENCSPUENV()

BasicGraphics Library (Chapter 7)

Functions added:

GetDrawEnv2()

Functions revised:

ClearImage()
ClearImage2()
SetDrawMode()

Extended Graphics Library (Chapter 9)

Functions revised:

GsCOORDINATE2()
GsGetLs()
GsGetLw()

Extended CD-ROM Library (Chapter 11)

Functions revised:

DsLastCom()

HMD Library (Chapter 17)

Functions revised

GsCOORDUNIT()
GsGetLwUnit()
GsGetLsUnit()

Related Documentation

This manual should be read in conjunction with the *Run-Time Library Overview*, since the *Overview* summarizes the use of the libraries.

Note: the Developer Support Web site posts current developments regarding the run-time libraries and also provides notice of future documentation releases and upgrades.

Manual Structure

The *Library Reference* contains nineteen chapters providing definitions of library structures and functions.

Generally, each chapter defines the structures and/or functions of a single library. Note, however, that some chapters provide definitions for several related libraries. In particular, note that Chapter 2, the Standard C Library, describes libc and libc2. Chapter 12, the Controller/Peripherals Library, describes libetc, libgun, libpad and libtap.

Developer Reference Series

This manual is part of the *Developer Reference Series*, a series of technical reference volumes covering all aspects of PlayStation development. The complete series is listed below:

Manual	Description
PlayStation Hardware	Describes the PlayStation hardware architecture and overviews its subsystems.
PlayStation Operating System	Describes the PlayStation operating system and related programming fundamentals.
Run-Time Library Overview	Describes the structure and purpose of the run-time libraries provided for PlayStation software development.
Run-Time Library Reference	Defines all available PlayStation run-time library functions, macros and structures.
Inline Programming Reference	Describes in-line programming using DMPSX, GTE inline macro and GTE register information.
SDevTC Development Environment	Describes the SDevTC (formerly "Psy-Q") Development Environment for PlayStation software development.
3D Graphics Tools	Describes how to use the PlayStation 3D Graphics Tools, including the animation and material editors.
Sprite Editor	Describes the Sprite Editor tool for creating sprite data and background picture components.
Sound Artist Tool	Provides installation and operation instructions for the DTL-H800 Sound Artist Board and explains how to use the Sound Artist Tool software.
File Formats	Describes all native PlayStation data formats.
Data Conversion Utilities	Describes all available PlayStation data conversion utilities, including both stand-alone and plug-in programs.

Manual	Description
CD Emulator	Provides installation and operation instructions for the CD Emulator subsystem and related software.
CD-ROM Generator	Describes how to use the CD-ROM Generator software to write CD-R discs.
Performance Analyzer User Guide	Provides general instructions for using the Performance Analyzer software.
Performance Analyzer Technical Reference	Describes how to measure software performance and interpret the results using the Performance Analyzer.
DTL-H2000 Installation and Operation	Provides installation and operation instructions for the DTL-H2000 Development System.
DTL-H2500/2700 Installation and Operation	Provides installation and operation instructions for the DTL-H2500/H2700 Development Systems.

Typographic Conventions

Certain Typographic Conventions are used through out this manual to clarify the meaning of the text. The following conventions apply:

Convention	Meaning
<i>Italic</i>	Function arguments and structure members.
<code>Courier</code>	Literal program code.
Medium Bold	Types and structure/function names (in structure/function definitions only)
Blue	Hyperlink to function or structure description

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
<i>In North America:</i>	<i>In North America:</i>
Attn: Developer Tools Coordinator	E-mail: DevTech_Support@playstation.sony.com
Sony Computer Entertainment America	Web: http://www.scea.sony.com/dev
919 East Hillsdale Blvd., 2nd floor	Developer Support Hotline:
Foster City, CA 94404	(650) 655-8181
Tel: (650) 655-8000	(Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
<i>In Europe:</i>	<i>In Europe:</i>
Attn: Production Coordinator	E-mail: dev_support@playstation.co.uk
Sony Computer Entertainment Europe	Web: https://www-s.playstation.co.uk

Order Information	Developer Support
Waverley House 7-12 Noel Street London W1V 4HH Tel: +44 (0) 171 447 1600	Developer Support Hotline: +44 (0) 171 447 1680 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT or BST/BDT)

Chapter 1: Kernel Library

Table of Contents

Structures

DIRENTRY	1-3
EvCB	1-4
EXEC	1-5
TCB	1-6
TCBH	1-7
ToT	1-8

Functions

calloc2	1-9
calloc3	1-10
cd	1-11
ChangeClearPAD	1-12
ChangeTh	1-13
close	1-14
CloseEvent	1-15
CloseTh	1-16
DeliverEvent	1-17
DisableEvent	1-18
DisablePAD	1-19
EnableEvent	1-20
EnablePAD	1-21
EnterCriticalSection	1-22
erase	1-23
Exception	1-24
Exec	1-25
ExitCriticalSection	1-26
firstfile	1-27
FlushCache	1-28
format	1-29
free2	1-30
free3	1-31
GetConf	1-32
GetCr	1-33
GetGp	1-34
GetRCnt	1-35
GetSp	1-36
GetSr	1-37
GetSysSp	1-38
InitHeap	1-39
InitHeap2	1-40
InitHeap3	1-41
InitPAD	1-42
ioctl	1-43
Krom2RawAdd	1-44
Krom2RawAdd2	1-45
Load	1-46
LoadExec	1-47
LoadTest	1-48

lseek	1-49
malloc2	1-50
malloc3	1-51
nextfile	1-52
open	1-53
OpenEvent	1-54
OpenTh	1-55
read	1-56
realloc2	1-57
realloc3	1-58
rename	1-59
ResetRCnt	1-60
ReturnFromException	1-61
SetConf	1-62
SetMem	1-63
SetRCnt	1-64
SetSp	1-65
StartPAD	1-66
StartRCnt	1-67
StopPAD	1-68
StopRCnt	1-69
SwEnterCriticalSection	1-70
SwExitCriticalSection	1-71
SystemError	1-72
TestEvent	1-73
undelele	1-74
UnDeliverEvent	1-75
WaitEvent	1-76
write	1-77
_96_init	1-78
_96_remove	1-79
_boot	1-80
_get_errno	1-81
_get_error	1-82

Structures

DIRENTRY

Directory entries.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>kernel.h</i>	2.x	12/14/98

Structure

```
struct DIRENTRY {
    char name[20];           Filename
    long attr;               Attributes (dependent on file system)
    long size;               File size (in bytes)
    struct DIRENTRY *next;   Pointer to next file entry (for user)
    char system[8];          Reserved by system
};
```

Explanation

Stores information relating to files registered in the file system.

See also

[firstfile\(\)](#), [nextfile\(\)](#)

EvCB

Event Control Block

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>kernel.h</i>	2.x	12/14/98

Structure

```

struct EvCB {
    u_long desc;           Cause descriptor
    long status;          Status
    long spec;            Event type
    long mode;            Mode
    (long *FHandler);     Pointer to a function type handler
    long system[2];       Reserved by system
};

```

Explanation

Stores information for each event.

See also[OpenEvent\(\)](#), [GetConf\(\)](#), [SetConf\(\)](#).

EXEC

Execution file data structure.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>kernel.h</i>	2.x	12/14/98

Structure

```
struct EXEC {
    unsigned long pc0;           Execution start address
    unsigned long gp0;           gp register initial value
    unsigned long t_addr;        Starting address of initialized text section
    unsigned long t_size;        Size of text section
    unsigned long d_addr;        Starting address of initialized data section
    unsigned long d_size;        Size of initialized data section
    unsigned long b_addr;        Uninitialized data section start address
    unsigned long b_size;        Uninitialized data section size
    unsigned long s_addr;        Stack start address (specified by the user)
    unsigned long s_size;        Stack size (specified by the user)
    unsigned long sp;            Register shunt variable
    unsigned long fp;            Register shunt variable
    unsigned long gp;            Register shunt variable
    unsigned long ret;           Register shunt variable
    unsigned long base;          Register shunt variable
};
```

Explanation

Stores information for loading and executing a program. The data is stored in the first 2K bytes of the execution file (PS-X EXE format). By adding stack information and transferring it to Exec(), the program is activated.

See also

[Exec\(\)](#)

TCB

Task Control Block.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>kernel.h</i>	2.x	12/14/98

Structure

```

struct TCB {
    long status;           Status
    long mode;           Mode
    unsigned long reg[NREGS]; Register saving area (specified by register designation
                                macro)
    long system[6];      Reserved by system
};

```

Explanation

Stores a context (including contents of the registers) for thread management.

See also[OpenTh\(\)](#), [ChangeTh\(\)](#), [GetConf\(\)](#), [SetConf\(\)](#)

TCBH

Task Control Block Header.

Library	Header File	Introduced	Documentation Date
libapi.lib	kernel.h	2.x	12/14/98

Structure

```
struct TCBH {
    struct TCB *entry;           Pointer to execution TCB
    long flag;                   System reserved
};
```

Explanation

Used for thread management. *entry* is a pointer to the currently executing TCB.

See also

[OpenTh\(\)](#), [ChangeTh\(\)](#)

ToT

System Table Information.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>kernel.h</i>	<i>2.x</i>	<i>12/14/98</i>

Structure

```

struct ToT {
    unsigned long *head;           Pointer to a system table start address
    long size;                   System table size (in bytes)
};

```

Explanation

Information about various system tables used by the kernel. The tables begin at address 0x00000100.

Functions

calloc2

Allocate a block in main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	12/14/98

Syntax

```
void *calloc2(
size_t n,           Number of partitions
size_t s)          Size of one partition
```

Explanation

Allocates a block of n*s bytes in the heap memory and initializes it to 0. Corresponds to InitHeap2().

Return value

Pointer to the allocated memory block. If allocation fails, NULL is returned.

See also

[InitHeap2\(\)](#), [malloc2\(\)](#), [realloc2\(\)](#), [free2\(\)](#)

calloc3

Allocate a block in main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	12/14/98

Syntax

```
void *calloc3 (  
size_t n,           Number of partitions  
size_t s)           Size of one partition
```

Explanation

Allocates a block of n*s bytes in the heap memory and initializes it to 0. Corresponds to InitHeap3().

Return value

A pointer to the allocated memory block. If allocation fails, NULL is returned.

See also

[InitHeap3\(\)](#), [malloc3\(\)](#), [realloc3\(\)](#), [free3\(\)](#)

ChangeClearPAD

Set the control driver.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
void ChangeClearPAD(
long val)                Vertical retrace line interruption clear flag
```

Explanation

if *val* is 1, interrupt processing in a control driver started by a vertical retrace line interrupt is completed. If *val* is 0, processing is passed to a lower priority interrupt module without completion.

See also

[StartPAD\(\)](#), [StopPAD\(\)](#), [StartCARD\(\)](#) (see libcard), [StopCARD\(\)](#) (see libcard)

ChangeTh

Change the thread to be executed.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long ChangeTh(
  unsigned long thread)           Thread descriptor
```

Explanation

Transfers execution to the thread specified by *thread*. The current thread is saved in a [TCB](#). This function returns when the original thread is restored.

Before executing ChangeTh(), initialize TCB reg [R-SR] to the following:

- The interrupt context is 0X404
- The main flow is 0X401

Return value

On success and re-execution, the function returns 1. On failure, it returns 0. The return value on re-execution can be changed by any other thread.

See also

[OpenTh\(\)](#)

close

Close a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
int close(  
int fd)
```

File descriptor

Explanation

Closes the file specified by *fd*.

Return value

fd, if the function succeeds, -1 otherwise.

See also

[Open\(\)](#)

CloseEvent

Close an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long CloseEvent(  
    unsigned long event)
```

Event descriptor

Explanation

Releases the [EvCB](#) specified by *event*. Must be executed in a critical section.

Return value

1 on success, 0 on failure.

See also

[OpenEvent\(\)](#), [EnterCriticalSection\(\)](#), [SwEnterCriticalSection\(\)](#)

CloseTh

Close a thread.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long closeTh(
unsigned long *thread*) Thread descriptor

Explanation

Closes a thread and releases its [TCB](#). Must be executed in a critical section.

Return value

1 on success, 0 on failure.

See also

[OpenTh\(\)](#), [EnterCriticalSection\(\)](#), [SwEnterCriticalSection\(\)](#)

DeliverEvent

Generate an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
void DeliverEvent(  
  unsigned long ev1,           Cause descriptor  
  long ev2)                    Event class
```

Explanation

Delivers an event if the event's current status is EvStACTIVE (event not yet generated, generation possible). If the event mode is EvMdlINTR, the event handler function is called. If the event mode is EvMdNOINTR, the event status is changed to EvStALREADY (event already occurred, generation prohibited). This function must be executed in a critical section.

See also

[UnDeliverEvent\(\)](#), [OpenEvent\(\)](#), [TestEvent\(\)](#), [EnterCriticalSection\(\)](#), [SwEnterCriticalSection\(\)](#), [DisableEvent\(\)](#), [EnableEvent\(\)](#), [WaitEvent\(\)](#), [CloseEvent\(\)](#)

DisableEvent

Disable an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long DisableEvent(  
unsigned long event)           Event descriptor
```

Explanation

Inhibits occurrence of an event specified by the descriptor *event*. It changes the event status to EvStWAIT (event generation prohibited).

Return value

1 on success, 0 on failure.

See also

[EnableEvent\(\)](#)

DisablePAD

Disable communication with the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void DisablePad(void)

Explanation

Temporarily disables communication with the controller.

Unlike StopPAD(), which deletes the controller handler activated by Vsync interrupts, this function simply skips controller communication by setting a flag in the handler.

Since a controller normally communicates via Vsync interrupts, this function can be used in situations when the controller status is needed less frequently than every 1/60 sec.

See also

[EnablePAD\(\)](#), [StopPAD\(\)](#)

EnableEvent

Enable occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long EnableEvent(  
unsigned long event)           Event descriptor
```

Explanation

Enables occurrence of an event specified by the descriptor *event*. It changes the event status to EvStACTIVE (event not yet generated, generation possible).

Return value

1 on success, 0 on failure.

See also

[DisableEvent\(\)](#), [TestEvent\(\)](#)

EnablePAD

Enable communication with the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void EnablePAD(*void*)

Explanation

Enables communication with a controller which was disabled with DisablePAD(). Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec. when the controller status is not needed.

See also

[DisablePAD\(\)](#)

EnterCriticalSection

Disable interrupts.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void EnterCriticalSection(void)

Explanation

Disables interrupts (enters a critical section).

Executes an internal system call and destroys the interrupt context. However, does not call the main function from the event handler callback interrupt context.

Return value

0 when this function is called in a critical section, 1 otherwise.

See also

[ExitCriticalSection\(\)](#)

erase

Delete a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

**long erase(
char **name*)** Pointer to a filename

Explanation

Deletes the file specified by *name*.

This function was formerly called “delete.”

Return value

1 on success, 0 on failure.

See also

[undelete\(\)](#)

Exception

Cause an interrupt.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void Exception(*void*)

Explanation

Causes an interrupt, and stores the current context in the execution [TCB](#). It is also valid in a critical section. Executes an internal call and destroys the exception context.

See also

[ChangeTh\(\)](#), [ReturnFromException\(\)](#)

Exec

Execute an execution file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long Exec(  
struct EXEC *exec,           Pointer to execution file information  
long argc,                  Number of arguments  
char *argv)                 Pointer to argument
```

Explanation

Executes a module that has already been loaded into memory, using the execution file information specified by `exec`. If `exec->s_addr` is 0, neither the stack nor frame pointers are set.

The function does the following:

- Clears a data section without initial values to zero.
- Saves `sp`, `fp`, and `gp`, and then initializes them. (`fp` is set to the same value as `sp`.)
- Sets the arguments of `main()` in the `a0` and `a1` registers.
- Calls the execution start address.
- Restores `sp`, `fp`, and `gp` after a return is made.

It must be executed in a critical section.

This function needs the ISO 9660 file system to run properly. Call `_96_init()` to initialize the system and `_96_remove()` to exit the system.

Return value

1 on success; 0 on failure.

See also

[Load\(\)](#), [_96_init\(\)](#), [_96_remove\(\)](#)

ExitCriticalSection

Enable interrupts.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void ExitCriticalSection(void)

Explanation

Enables interrupts (exits from a critical section).

Executes an internal system call and destroys the interrupt context. However, it does not call the main function from the event handler callback interrupt context.

See also

[EnterCriticalSection\(\)](#)

firstfile

Find the first file matching a filename.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
struct DIRENTRY *firstfile(  
char *name,           Pointer to a filename  
struct DIRENTRY *dir)  Pointer to the buffer holding information relating to the  
                        referenced file.
```

Explanation

Finds the first file corresponding to the filename pattern *name*, and stores data relating to this file in the directory *dir*. The wildcard characters “?” (standing for any one character) and “*” (standing for a character string of any length) can be used in the filename pattern. Characters specified after “*” are ignored.

Return value

Returns *dir* if it succeeds, and 0 otherwise.

See also

[nextfile\(\)](#)

FlushCache

Flush instruction cache.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	9/1/99

Syntax

void FlushCache(void)

Explanation

Flushes the instruction cache (I-cache). Must be executed in a critical section.

Because this function can hang if it is called during DMA transfer, it must be called after confirming that DMA transfer is complete.

See also

[EnterCriticalSection\(\)](#)

format

Initialize file system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long format(

char *fs)

Pointer to file system name

Explanation

Initializes file system *fs*. This function is only effective on writeable file systems.

When initializing the Memory Card, it's preferable to use the libcard function `_card_format()`.

Return value

Always returns 1.

See also

`_card_format()` (see libcard)

free2

Free allocated memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	12/14/98

Syntax

void free2

(void **block*)

Area to be released

Explanation

Releases a memory block that was allocated by `calloc2()`, `malloc2()`, or `realloc2()`. Corresponds to `InitHeap2()`.

See also

InitHeap2(), calloc2(), malloc2(), realloc2()

GetConf

Get the kernel configuration.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
void GetConf(
  unsigned long *ev,      Pointer to number of event management blocks
  unsigned long *tcb,     Pointer to number of task management blocks
  unsigned long *sp)      Ignored
```

Explanation

Stores a system configuration parameter set by SetConf() to the address given by the pointer as the argument. It returns an undefined value before the execution of SetConf() because this function refers to its internal parameter.

See also

[SetConf\(\)](#)

GetCr

Get cause register value.

Library	Header File	Introduced	Documentation Date
libapi.lib	libapi.h	3.0	12/14/98

Syntax

unsigned long GetCr(void)

Explanation

Gets the value of the cause register (a coprocessor control register).

Table 1-1: Description of Cause-Register Bits for GetCr

Bit	Description
31-6	Reserved by the system
5-2	Exception code
	0000 External interrupt
	0001 Not used
	0010 Not used
	0011 Not used
	0100 Address read error
	0101 Address write error
	0110 Command bus error
	0111 Data bus error
	1000 System call
	1001 Break point
	1010 Undefined command
	1011 Co-processor not mounted
	1100 Overflow
1-0	Reserved by the system

Return value

The current cause register value.

See also

[OpenTh\(\)](#)

GetGp

Get value of gp register.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

unsigned long GetGp(*void*)

Explanation

Gets the value of the gp register.

Return value

The current gp register value.

See also

[OpenTh\(\)](#), [Load\(\)](#), [Exec\(\)](#)

GetRCnt

Get value of a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long GetRCnt(
long spec)           Root counter
```

Explanation

Returns the current value of root counter *spec*. To be used when root counter *spec* has been set by SetRCnt to a polling mode (RCntMdNOINTR).

Return value

The 32-bit unsigned expanded counter value. On failure, it returns -1.

See also

[SetRCnt\(\)](#), [StartRCnt\(\)](#), [StopRCnt\(\)](#), [ResetRCnt\(\)](#)

GetSp

Get value of stack pointer.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

unsigned long GetSp(*void*)

Explanation

Gets value of sp register.

Return value

The current sp register value.

See also

[OpenTh\(\)](#), [Load\(\)](#), [Exec\(\)](#), [SetSp\(\)](#)

GetSr

Get value of status register.

Library	Header File	Introduced	Documentation Date
libapi.lib	libapi.h	3.0	12/14/98

Syntax

unsigned long GetSr(void)

Explanation

Gets the value of the status register.

Table 1-2: Description of Status-Register Bits for GetSr

Bit	Description
31-28	Co-processor installation flag (1: Installed); Bit 29 is GTE.
27-11	Reserved by the system
10	Always 1
9-3	Reserved by the system
2	Main flow interrupt permission (1: Permission)
1	Reserved by the system
0	Interrupt permission (1: Permission)

Return value

The current status register value.

See also

[OpenTh\(\)](#)

GetSysSp

Get address of system stack.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

long GetSysSp(void)

Explanation

Gets the highest address of a system stack area for event handler function execution.

The size of the stack area is 2 K-bytes.

Return value

Highest address of the system stack area

See also

[GetSp\(\)](#)

InitHeap

Initialize heap area.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.0	12/14/98

Syntax

```
void InitHeap(
  unsigned long *head,      Pointer to heap start address
  unsigned long size)       Heap size (a multiple of 4, in bytes)
```

Explanation

Initializes a group of standard function library memory control functions. After using this function, malloc(), free(), etc. are usable.

There is some overhead, so the entire *size* in bytes cannot be used.

Must be executed in a critical section. If several executions of this function overlap, the previous memory control information is lost.

See also

[malloc\(\)](#) (see libc/libc2)

InitHeap2

Initialize heap area.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	12/14/98

Syntax

```
void InitHeap(
void *head,           Pointer to heap start address
long size)            Heap size (a multiple of 4, in bytes)
```

Explanation

Initializes a heap area of *size* bytes. (Since there is overhead, the entire size in bytes cannot be used.)

After calling this function, the library memory routines in the "malloc3" group (malloc3(), free3(), etc.) are usable. This routine fixes a bug in InitHeap() but has larger program size since this is a memory resident function. See "Memory Allocation Functions" in the Kernel chapter of the Library Overview for more information on the malloc systems.

If several executions of this function overlap, the previous memory control information is lost.

See also

[InitHeap\(\)](#), [malloc2\(\)](#), [realloc2\(\)](#), [calloc2\(\)](#), [free2\(\)](#)

InitPAD

Initialize the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long InitPAD(
char *bufA, char *bufB,      Pointers to incoming data buffers
long lenA, long lenB)       Length of incoming data buffers (in bytes)
```

Explanation

Registers a receive data buffer for the controller. For the format of this buffer, see “Receive Buffer Data Format” of Chapter 13 (Controller/Peripherals Library) of the Library Overview.

Since it is possible for mistakes to occur when an unexpected controller is connected to the receive data length, always allocate 34 bytes.

When using the Multi Tap, use InitTAP(). When using the gun controller, use InitGUN().

Return value

Always 1.

See also

[StartPAD\(\)](#), [StopPAD\(\)](#), [ChangeClearPAD\(\)](#), [InitTAP\(\)](#) (see libetc), [InitGUN\(\)](#) (see libetc).

ioctl

Control devices.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long ioctl(
int *fd*,
int *com*,
int *arg*)

File descriptor
Control command
Control command argument

Explanation

Executes control commands on the device. Details of the commands and their arguments are given separately for each device.

Return value

1 if it succeeds and 0 otherwise.

See also

[open\(\)](#)

Krom2RawAdd

Get Kanji font pattern addresses.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
unsigned long Krom2RawAdd(
  unsigned short sjiscode)      Shift-JIS code
```

Explanation

Gets the starting address in the kernel of the font pattern for the Kanji character specified by *sjiscode*.

Refer to the codeview sample in \psx\kanji\sjiscode for a list of usable fonts and the actual fonts themselves.

Return value

The starting address of a Kanji font pattern. If there is no font data corresponding to the specified Kanji character, a value of -1 is returned.

Bug alert: The normal arguments are Shift-JIS code values between 0x8140~0x84BE or 0x889F~0x9872. If a Shift-JIS code within that region corresponds to a blank area in the code table, a font pattern unrelated to that code is returned as the starting address. This problem has been corrected in Krom2RawAdd2, so we recommend using Krom2RawAdd2 to obtain the font pattern starting address.

See also

[Krom2RawAdd2\(\)](#)

Krom2RawAdd2

Get shift-JIS font pattern addresses.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.2	12/14/98

Syntax

```
unsigned long Krom2RawAdd2(
  unsigned short sjiscode)      Shift-JIS code
```

Explanation

Gets the starting address in the font pattern kernel for the non-Kanji/Kanji level 1 character specified by the *sjiscode*.

(Refer to the codeview sample in \psx\kanji\sjiscode for a list of usable fonts and the actual fonts themselves.)

Return value

The font pattern starting address. When there is no font data corresponding to the specified shift-JIS code, an address containing a full space font pattern is returned.

See also

[Krom2RawAdd\(\)](#)

Load

Load an execution file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long Load(
char *name,           Pointer to filename
struct EXEC *exec)    Pointer to execution file information
```

Explanation

Reads the PlayStation EXE format file *name* to the address specified by its internal header, and writes internal information to *exec*.

This function needs the ISO 9660 file system to run properly. To initialize this system, call `_96_init()`; to exit the system, call `_96_remove()`.

Calls `FlushCache()` internally.

Return value

1 if it succeeds, and 0 otherwise.

See also

[Exec\(\)](#), [FlushCache\(\)](#), [_96_init\(\)](#), [_96_remove\(\)](#)

LoadExec

Load and execute an execution file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
void LoadExec(  
char *name,           Pointer to a PS-X EXE format execution file name (fewer than 19  
                      characters)  
unsigned long s_addr,  Stack area starting address  
unsigned long s_size)  Number of bytes in stack area
```

Explanation

Calls Load() and Exec(), then reads a file name into memory and executes the file. *s_addr* and *s_size* are passed to Exec() and set by the structure EXEC.

This function needs the ISO 9660 file system to run properly. To initialize this system, call _96_init(); to exit the system, call _96_remove().

See also

[Load\(\)](#), [Exec\(\)](#), [_96_init\(\)](#), [_96_remove\(\)](#)

LoadTest

Load test.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long LoadTest(

char **name*,

Pointer to filename

struct EXEC **exec*)

Pointer to data in an execution file

Explanation

Writes internal information from a PS-X EXE format file *name* to *exec*.

Return value

The execution starting address. On failure, it returns 0.

See also

[Load\(\)](#)

Iseek

Move a file pointer.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

unsigned long Iseek(

int <i>fd</i>,	File descriptor
unsigned int <i>offset</i>,	Number of bytes to move pointer
int <i>flag</i>)	Start point flag

Explanation

Moves a file pointer of the device indicated by the descriptor *fd*.

If *flag* is SEEK_SET, movement starts at the start of the file; if *flag* is SEEK_CUR, movement starts with the current position.

This function does not apply to a PC communication link.

Return value

The current file pointer, if it succeeds. On failure, it returns -1.

See also

[open\(\)](#), [read\(\)](#), [write\(\)](#)

malloc2

Allocate main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	12/14/98

Syntax

```
void *malloc2 (
size_t s)                                Size of memory block to be allocated
```

Explanation

Allocates s bytes of memory block from the heap memory. InitHeap2() must be executed in advance.

Heap memory is defined as below:

Low Address	Module Highest Address + 4
High Address	On-board memory - 64KB

Return value

A pointer to the allocated memory block. On failure, NULL is returned.

See also

[InitHeap2\(\)](#), [calloc2\(\)](#), [realloc2\(\)](#), [free2\(\)](#)

malloc3

Allocate main memory.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	12/14/98

Syntax

```
void *malloc3 (
size_t s)                Size of memory block to be allocated
```

Explanation

Allocates s bytes of memory block from the heap memory. InitHeap3() must be executed in advance.

Refer to the section entitled "Memory Allocation Functions" in the Kernel chapter of the Library Overview for the differences between the various malloc systems.

Return value

A pointer to the allocated memory block. If allocation fails, NULL is returned.

See also

[InitHeap3\(\)](#), [calloc3\(\)](#), [realloc3\(\)](#), [free3\(\)](#)

nextfile

Find the next file matching a filename.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
struct DIRENTRY *nextfile(
struct DIRENTRY *dir)          Pointer to buffer holding file information
```

Explanation

Continues a file lookup under the same conditions as the previous call to `firstfile()`. If it finds a corresponding file, it stores file information in *dir*.

If the shell cover of the CD-ROM drive has been opened since the execution of the previous `firstfile()` call, this function fails, and reports that the file has not been found.

Return value

dir if it succeeds, and 0 otherwise.

See also

[firstfile\(\)](#)

open

Open a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long open(  
char *devname,           Pointer to a filename  
int flag)                Open mode
```

Explanation

Opens a device for low-level input/output, and returns the descriptor. The values of *flag* are dependent on the device; they can be the following:

Table 1-3: Open Modes

Macro	Open mode
O_RDONLY	Read only
O_WRONLY	Write only
O_RDWR	Both read and write
O_CREAT	Create new file
O_NOBUF	Non-buffer mode
O_NOWAIT	Asynchronous mode

Return value

The file descriptor, if the function succeeds. On failure, it returns -1.

See also

[close\(\)](#)

OpenEvent

Open an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long OpenEvent(
  unsigned long desc,      Cause descriptor
  long spec,              Event type
  long mode,              Mode
  long *func())           Pointer to the handler function
```

Explanation

Secures an [EvCB](#) for an event with the descriptor *desc* and event class *spec*. Must be executed in a critical section.

Return value

The event descriptor, if the function succeeds. On failure, it returns -1.

See also

[CloseEvent\(\)](#), [DeliverEvent\(\)](#), [EnterCriticalSection\(\)](#)

OpenTh

Open a thread.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

unsigned long **OpenTh**(
unsigned long (**func*)(),
unsigned long *sp*,
unsigned long *gp*)

Pointer to the execution start function
Stack pointer value
Global pointer value

Explanation

Secures a [TCB](#) for a given thread, and initializes it with the arguments given. Must be executed in a critical section.

The thread can be executed using [ChangeTh\(\)](#).

Return value

The thread descriptor, if the function succeeds. On failure, it returns -1.

See also

[CloseTh\(\)](#), [ChangeTh\(\)](#)

read

Read data from a file

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax**long read (****long** *fd*,**void** **buf*,**long** *n*)

File descriptor

Pointer to read buffer address

Number of bytes to read

ExplanationReads *n* bytes from the descriptor *fd* to the area specified by *buf*.**Return value**

The actual number of bytes read into the area. An error returns -1.

See also[open\(\)](#), [lseek\(\)](#)

realloc2

Change a block’s memory allocation.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	3.6	12/14/98

Syntax

void *realloc2(
void **block*,
size_t *s*)

Area to be reallocated
Size of area to be reallocated

Explanation

Changes the size of the memory block previously allocated to *s* bytes. Same as `malloc2()` when *block* is NULL. Corresponds to `InitHeap2()`.

Return value

A pointer to the reallocated memory block. The address may be from the original. If reallocation fails, NULL is returned, and the original block is not released.

See also

[calloc2\(\)](#), [malloc2\(\)](#), [free2\(\)](#)

realloc3

Change a block's memory allocation.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>malloc.h</i>	4.0	12/14/98

Syntax

void *realloc3

(**void ****block*,

size_t *s*)

Area to be reallocated

Size of area to be reallocated

Explanation

Changes the size of the memory block previously allocated to *s* bytes. When *block* is NULL, it operates in the same way as `malloc3()`.

Return value

A pointer to the reallocated block address; this address may be different from the original. If reallocation fails, NULL is returned, and the original block is not released. NULL is also returned if *s* is 0.

See also

[InitHeap3\(\)](#), [calloc3\(\)](#), [malloc3\(\)](#), [free3\(\)](#)

rename

Change a file name.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long rename(

char *src,

Pointer to the old filename

char *dest)

Pointer to the new filename

Explanation

Changes a filename from *src* to *dest*. In both cases, the full path from the device name must be specified. This function is only effective on writeable file systems.

Return value

1 if it succeeds, and 0 otherwise.

See also

[open\(\)](#)

ResetRCnt

Reset a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long ResetRCnt(
long *spec*) Root counter specification

Explanation

Resets the root counter *spec* to 0.

Return value

1 if it succeeds, and 0 otherwise. (0 is always returned if you specify RCntCNT3, since it can't be set.)

See also

[SetRCnt\(\)](#), [GetRCnt\(\)](#), [StartRCnt\(\)](#), [StopRCnt\(\)](#)

ReturnFromException

Return from exception.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void ReturnFromException(void)

Explanation

Accesses the exception context and returns from exception processing. It is used in an event handler or callback function.

See also

[Exception\(\)](#)

SetConf

Modify the kernel configuration.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long SetConf(
unsigned long *ev*, Number of event control block (EvCB) elements
unsigned long *tcb*, Number of task control block (TCB) elements
unsigned long *sp*) Ignored

Explanation

Modifies system configuration parameters. The contents of event and task control blocks, and all settings for event handlers and callback functions in each library, are destroyed. However, file descriptors are not affected (all the descriptors should be closed before SetConf() call) because most of the device drivers are driven by the event handler.

All patches to the kernel are invalidated.

This function should be executed at the head of the first execution file. The operations of libraries initialized before the execution of this function are not ensured.

This function eliminates the ISO-9660 file system installed in the kernel immediately after activation (call _96_init() to reinstate). The result of operations on the opened files are not predictable.

If the number of the designated elements exceeds the maximum, the operation of the system after the execution of this function is not defined.

Return value

1 if the function succeeds, 0 otherwise.

See also

[GetConf\(\)](#)

SetMem

Modify the valid memory size.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
void SetMem(
unsigned long n)           Valid memory size (in megabytes)
```

Explanation

Changes the valid memory size to *n*. It must be 2 or 8 (megabytes); any other values are ignored.

Memory access out of the valid range causes a CPU exception regardless of how much physical memory is present.

See also

[SetConf\(\)](#)

SetRCnt

Set a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long SetRCnt(
long spec,           Root counter specification
unsigned short target, Target value
long mode)           Mode
```

Explanation

Sets the root counter in *spec* to the target value in *target*, and the mode in *mode*.

Return value

1 if it succeeds, and 0 otherwise. (0 is always returned if you specify RCntCNT3, since it can't be set.)

See also

[GetRCnt\(\)](#), [StartRCnt\(\)](#), [StopRCnt\(\)](#), [ResetRCnt\(\)](#)

SetSp

Set the stack pointer.

Library	Header File	Introduced	Documentation Date
libapi.lib	libapi.h	2.x	12/14/98

Syntax

unsigned long SetSp(
unsigned long new-sp) value to set sp register

Explanation

Sets the sp register to the value new-sp.

Return value

The previous sp register value.

See also

[OpenTh\(\)](#), [Load\(\)](#), [Exec\(\)](#), [GetSp\(\)](#)

StartPAD

Start reading the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long StartPAD(*void*)

Explanation

Triggered by the interruption of a vertical retrace line, this function starts to read the controller. ChangeClearPAD (1) is executed internally.

Interrupts are permitted.

Return value

Always returns 1.

See also

[InitPAD\(\)](#), [ChangeClearPAD\(\)](#), [StopPAD\(\)](#)

StopPAD

Stop reading the controller.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void StopPAD(*void*)

Explanation

Stops reading the controller. Interrupts are not permitted.

See also

[InitPAD\(\)](#), [ChangeClearPAD\(\)](#), [StartPAD\(\)](#)

StopRCnt

Stop a root counter.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

```
long StopRCnt(
long spec)           Root counter
```

Explanation

Disables interrupts for root counter *spec*.

Return value

1 if it succeeds, and 0 otherwise. (0 is always returned if you specify RCntCNT3, since it can't be set.)

See also

[StartRCnt\(\)](#), [SetRCnt\(\)](#), [ResetRCnt\(\)](#), [GetRCnt\(\)](#)

SwEnterCriticalSection

Suppress interrupts.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void SwEnterCriticalSection(void)

Explanation

Suppresses interrupts. Because no system call interrupt is generated internally, this function can be invoked in event handling and callback functions. It must be executed in a critical section.

See also

[EnterCriticalSection\(\)](#), [SwExitCriticalSection\(\)](#)

SwExitCriticalSection

Enable interrupts.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void SwExitCriticalSection(void)

Explanation

Enables interrupts. Because no system call interrupt is generated internally, this function can be invoked in event handling and callback functions.

Must be executed in a critical section.

See also

[EnterCriticalSection\(\)](#), [SwExitCriticalSection\(\)](#)

SystemError

Display the system error screen.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void SystemError(

char *c*,

long *n*)

Error identification character (Alphabetic character)

Error identification code (0 to 999)

Explanation

Displays a detected system error for the user. On the PlayStation®, it calls `exit()`.

TestEvent

Test an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long TestEvent(
unsigned long *event*) Event descriptor

Explanation

Checks to see whether or not the event specified by the descriptor *event* has occurred. If so, the function restores the event state to EvStACTIVE.

Return value

1 if the event is found to have occurred, 0 otherwise.

See also

[DeliverEvent\(\)](#), [EnableEvent\(\)](#), [WaitEvent\(\)](#), [OpenEvent\(\)](#), [CloseEvent\(\)](#), [UnDeliverEvent\(\)](#), [DisableEvent\(\)](#)

undelete

Resurrect a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

**long undelete(
char **name*)** Pointer to filename

Explanation

Resurrects the previously deleted file specified by *name*.

Return value

1 if it succeeds, and 0 otherwise.

See also

[erase\(\)](#)

UnDeliverEvent

Cancel an event.

Library	Header File	Introduced	Documentation Date
libapi.lib	libapi.h	2.x	12/14/98

Syntax

void UnDeliverEvent(
unsigned long *ev1*, Cause descriptor
long *ev2*) Event class

Explanation

Returns event state from EvStALREADY (already occurred) to EvStACTIVE if the event mode is EvMdNOINTR. Must be executed in a critical section.

See also

[DeliverEvent\(\)](#), [EnableEvent\(\)](#), [OpenEvent\(\)](#), [TestEvent\(\)](#), [WaitEvent\(\)](#), [EnterCriticalSection\(\)](#)

WaitEvent

Wait for the occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

long WaitEvent(
unsigned long *event*) Event descriptor

Explanation

Waits until an event specified by the descriptor *event* occurs, and returns after restoring the event state to EvStACTIVE.

Return value

1 if it succeeds, and 0 otherwise.

See also

[TestEvent\(\)](#), [OpenEvent\(\)](#), [CloseEvent\(\)](#), [DeliverEvent\(\)](#), [UnDeliverEvent\(\)](#), [DisableEvent\(\)](#), [EnableEvent\(\)](#)

write

Write data to a file.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

**int write(
int *fd*,
char **buf*,
int *n*)**

File descriptor
Pointer to the write buffer address
Number of bytes to be written

Explanation

Writes *n* bytes from the descriptor *fd* to the area specified by *buf*.

Return value

The number of bytes actually written to the area. If there is an error, -1 is returned.

See also

[open\(\)](#), [lseek\(\)](#)

_96_init

Install the ISO-9660 file system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void _96_init(void)

Explanation

Installs the ISO-9660 file system driver that manages access to the CD-ROM.

See also

[_96_remove\(\)](#)

_96_remove

Remove the ISO-9660 file system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	2.x	12/14/98

Syntax

void _96_remove(void)

Explanation

Removes the ISO-9660 file system driver that manages access to the CD-ROM.

See also

[_96_init\(\)](#)

_boot

Reboot the system.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

void _boot(void)

Explanation

Reboots the system. This function is useful for demonstration programs; don't use it for general title applications.

_get_errno

Get the latest I/O error code.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

int _get_errno(void)

Explanation

Gets the latest error code from all file descriptors. (Error codes are defined in sys/errno.h.)

Return value

Error code.

See also

[_get_error\(\)](#)

get_error

Get an error code for a file descriptor.

Library	Header File	Introduced	Documentation Date
<i>libapi.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
int _get_error(  
int fd)
```

File descriptor

Explanation

Gets the most recent error code of the specified file descriptor. (Error codes are defined in sys/errno.h.)

Return value

Error code.

See also

[get_errno\(\)](#)

Chapter 2: Standard C Library

Table of Contents

Functions

abs	2-3
atoi	2-4
atol	2-5
bcmp	2-6
bcopy	2-7
bsearch	2-8
bzero	2-9
calloc	2-10
exit	2-11
free	2-12
getc	2-13
getchar	2-14
gets	2-15
isXXXX...	2-16
labs	2-17
longjmp	2-18
malloc	2-19
memchr	2-20
memcmp	2-21
memcpy	2-22
memmove	2-23
memset	2-24
printf	2-25
putc	2-26
putchar	2-27
puts	2-28
qsort	2-29
rand	2-30
realloc	2-31
setjmp	2-32
sprintf	2-33
srand	2-34
strcat	2-35
strchr	2-36
strcmp	2-37
strcpy	2-38
strcspn	2-39
strlen	2-40
strncat	2-41
strncmp	2-42
strncpy	2-43
strpbrk	2-44
strrchr	2-45
strspn	2-46
strstr	2-47
strtok	2-48
strtol	2-49
strtoul	2-50
toascii	2-51
tolower	2-52
toupper	2-53

abs

Calculate absolute value.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>abs.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
int abs(  
int i)                Integer
```

Explanation

Calculates the absolute value of the integer *i*. On the R3000, int and long are the same size, so this function is equivalent to labs().

Return value

Absolute value of the argument.

See also

[labs\(\)](#)

atoi

Convert a string to an integer.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	2.x	12/14/98

Syntax

```
int atoi(  
char *s)           Pointer to a character string
```

Explanation

Converts a string to its integer equivalent. On this system, it is equivalent to `atol()`.

Return value

Integer equivalent of `s`.

See also

[atol\(\)](#), [strtol\(\)](#), [strtoul\(\)](#)

atol

Convert a character string to a long.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	2.x	12/14/98

Syntax

**long atol(
char *s)** Pointer to a character string

Explanation

Converts a string to its long equivalent. On this system, it is equivalent to `atoi()`.

Return Value

Integer equivalent of `s`.

See also

[atoi\(\)](#), [strtol\(\)](#), [strtoul](#)

bcmp

Compare memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
int bcmp(  
u_char *b1,           Pointer to first block  
u_char *b2,           Pointer to second block  
int n)                Number of bytes to be compared
```

Explanation

Compares the first *n* bytes of *b1* and *b2*.

Return value

0 if *b1* == *b2*.

<0 if *b1* < *b2*

>0 if *b1* > *b2*.

See also

[memcmp\(\)](#)

bcopy

Copy a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
void bcopy(  
u_char *src,           Pointer to copy source  
u_char *dest,          Pointer to copy destination  
int n)                 Number of bytes copied
```

Explanation

Copies the first *n* bytes of *src* to *dest*.

See also

[memcpy\(\)](#), [memmove\(\)](#)

bsearch

Perform a binary search.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdlib.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
void *bsearch(  
u_char *key,           Pointer to the value to be searched for  
u_char *base,          Pointer to the array to be searched  
size_t n,              Number of elements  
size_t w,              Size of one element  
int (*fcmp)())          Pointer to address of comparison function
```

Explanation

Carries out a binary search on a table of *n* items (of item size *w*) starting from *base*, for an item matching *key*.

Return value

The address of the first item matching the search key. If no matching item is found, 0 is returned.

bzero

Fill a memory block with zeros.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
void bzero(  
u_char *p,           Pointer to memory block  
int n)              Size
```

Explanation

Sets *n* bytes to the value 0, starting from *p*..

calloc

Allocate main memory.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	2.x	12/14/98

Syntax

```
void *calloc(  
size_t n,           Number of blocks  
size_t s)           Size of block
```

Explanation

Allocates a memory area of *n* blocks of *s* bytes each from the heap and initializes it to 0.

Return value

A pointer to the memory area allocated. If the function fails, it returns NULL.

See also

[malloc\(\)](#), [realloc\(\)](#), [free\(\)](#)

exit

Terminate a program normally.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdlib.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
void exit(  
int err)           Error code
```

Explanation

When executed on the PlayStation®, a system error notice window is displayed (including the error code), and the system enters an infinite loop. When executed on a development machine, the program currently being executed is terminated, and the system returns to the debug monitor.

free

Release an allocated memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	2.x	12/14/98

Syntax

void free(

void **block*)

Pointer to a memory block allocated by a function such as malloc().

Explanation

Releases a memory block that was allocated by calloc(), malloc() or realloc().

See also

[calloc\(\)](#), [malloc\(\)](#), [realloc\(\)](#)

getc

Get one character from a stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
char getc(
int fd)           File descriptor
```

Explanation

Gets one character from the file indicated by *fd*.

Devices and systems with a block size of 1 may all be used as the standard input/output stream as follows:

- Close (0);
- Close (1);
- Open (<device name>, O_RDONLY);
- Open (<device name>, O_WRONLY);

Return value

The character read. If the end of file is reached, or when an error is generated, the function returns EOF.

See also

[getchar\(\)](#), [gets\(\)](#), [putc\(\)](#)

getchar

Get one character from the standard input stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

char **getchar**(*void*)

Explanation

Gets one character from the standard input stream. It is the same as `getc(stdin)`.

Return value

The character read. If the end of file is reached, or when an error is generated, the function returns EOF.

See also

[getc\(\)](#), [gets\(\)](#), [putchar\(\)](#)

gets

Read a character string from standard input.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	2.x	12/14/98

Syntax

char *gets(

char *s)

Pointer to storage destination for input character string

Explanation

Reads a character string from the standard input stream (stdin) and stores it in s until a new-line character is read.

Return value

If this function succeeds, it returns s. The new-line character is discarded and a null character is written immediately after the last character read. If it reaches the end of the file, or if an error is generated, it returns NULL.

See also

[getc\(\)](#), [getchar\(\)](#), [puts\(\)](#)

isXXXX...

Test characters.

Library	Header File	Introduced	Documentation Date
libc\libc2.lib	ctype.h	2.x	12/14/98

Syntax

isXXXX(c) Character

Explanation

These are macros that perform the following tests on the character *c*:

Table 2-1: Character Macros

Name	Conditions
isalnum(c)	isalpha(c) isdigit(c)
isalpha(c)	isupper(c) islower(c)
isascii(c)	ASCII character
iscntrl(c)	Control character
isdigit(c)	Decimal
isgraph(c)	Printing characters other than space
islower(c)	Lower-case character
isprint(c)	Printing characters including space
ispunct(c)	Printing characters other than space and alphanumerics
isspace(c)	Space, new page, new line, restore, tab
isupper(c)	Upper-case character
isxdigit(c)	Hexadecimal

Return value

Non-zero if the character *c* satisfies the test conditions; 0 otherwise.

See also

[toascii\(\)](#), [tolower\(\)](#), [toupper\(\)](#)

labs

Calculate absolute value.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	2.x	12/14/98

Syntax

```
long labs(  
long i)           Long value
```

Explanation

Calculates the absolute value of *i*.

Return value

Absolute value of the argument.

See also

[abs\(\)](#)

longjmp

Non-local jump.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>setjmp.h</i>	2.x	12/14/98

Syntax

```
void longjmp(  
jmp_buf p,           Environment storage variable.  
int val)             setjmp() Return value
```

Explanation

Makes a non-local jump to the destination specified by *p*.

See also

[setjmp\(\)](#)

malloc

Allocate main memory.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	2.x	12/14/98

Syntax

```
void *malloc(
size_t s)           Number of bytes to be allocated
```

Explanation

Allocates a block of *s* bytes from the memory heap.

Note that the memory heap is defined as follows when the user program is activated:

Bottom address: top address of module + 4.

Top address: available memory -32KB.

This function has a bug whereby the area is not completely released in `free()`. This function can be replaced by `malloc2()` or `malloc3()` from `libapi`. For more information, refer to the Kernel Library chapter of the *Run-Time Library Overview*.

Return value

A pointer to the secured memory block. If allocation fails, NULL is returned.

See also

[calloc\(\)](#), [realloc\(\)](#), [free\(\)](#)

memchr

Search a memory block for a character.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
void *memchr(  
u_char *s,           Pointer to memory block  
u_char c,           Character  
int n)              Number of bytes
```

Explanation

Searches the memory block of n bytes starting from s , looking for the first appearance of the character c .

Return value

A pointer to the location at which c was found. If c was not found, NULL is returned.

See also

[strchr\(\)](#)

memcmp

Compare memory blocks.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
void *memcmp(  
u_char *s1,           Pointer to first memory block  
u_char *s2,           Pointer to second memory block  
int n)                Number of bytes to be compared
```

Explanation

Compares the first *n* bytes of *s1* and *s2*.

Return value

0 if *s1* = *s2*.
<0 if *s1* < *s2*.
>0 if *s1* > *s2*.

See also

[bcmp\(\)](#)

memcpy

Copy a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
void *memcpy(
u_char *dest,      Pointer to copy destination memory block
u_char *src,       Pointer to copy source memory block
int n)            Number of bytes copied
```

Explanation

Copies the first *n* bytes of *src* to *dest*.

Return value

Pointer to destination (*dest*).

See also

[bcopy\(\)](#)

memmove

Copy a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	2.x	12/14/98

Syntax

```
void *memmove(  
  u_char *dest,      Pointer to copy destination memory block  
  u_char *src,        Pointer to copy source memory block  
  int n)             Number of bytes copied
```

Explanation

Copies the first *n* bytes of *src* to *dest*. The block is copied correctly, even between overlapping objects.

Return value

Pointer to destination (*dest*).

See also

[bcopy\(\)](#), [memcpy\(\)](#)

memset

Write a character to a memory block.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>memory.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
void *memset(  
u_char *s,           Pointer to memory block  
u_char c,            Character  
int n)              Number of characters
```

Explanation

Writes *c* to the first *n* bytes of *s*.

Return value

Pointer to block (*s*).

printf

Print formatted output.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	2.x	2/24/99

Syntax

int printf(
char *fmt[, argument ...]) . Pointer to input format character string

Explanation

See a C language reference. Conversion directives f, e, E, g and G cannot be used. Use printf2() from libmath for floating-point representations.

Note: When printf() (or printf2(), putchar(), or puts()) is being executed in the main flow, and an interrupt occurs, text corruption or a hang-up can result if printf() is called during the interrupt. Therefore, pay attention to the call timing when calling printf() in an interrupt.

Return value

The length of the output character string. If an error is generated, the function returns NULL.

See also

[sprintf\(\)](#)

putc

Output one character to a stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	2.x	12/14/98

Syntax

void putc(

char *c*,

int *fd*)

Output character

File descriptor

Explanation

Outputs a character *c* to the file indicated by *fd*.

Return value

c if the function succeeds; EOF if an error is generated.

See also

[getc\(\)](#), [putchar\(\)](#), [puts\(\)](#)

putchar

Output one character to the standard output stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	2.x	2/24/99

Syntax

```
void putchar(  
char c)           Output character
```

Explanation

Outputs a character *c* to standard output. It is the same as `putc(stdout)`.

Note: When `printf()` (or `printf2()`, `putchar()`, or `puts()`) is being executed in the main flow, and an interrupt occurs, text corruption or a hang-up can result if `printf()` is called during the interrupt. Therefore, pay attention to the call timing when calling `printf()` in an interrupt.

Return value

c if the function succeeds; EOF if an error is generated.

See also

[getchar\(\)](#), [putc\(\)](#), [puts\(\)](#)

puts

Output a character string to the standard output stream.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>stdio.h</i>	<i>2.x</i>	<i>2/24/99</i>

Syntax

```
void puts(  
char *s)           Pointer to output character string
```

Explanation

Outputs a character string ending in NULL to the standard output stream (stdout), and finally outputs a newline character.

Note: When printf() (or printf2(), putchar(), or puts()) is being executed in the main flow, and an interrupt occurs, text corruption or a hang-up can result if printf() is called during the interrupt. Therefore, pay attention to the call timing when calling printf() in an interrupt.

Return value

A non-negative value, if the function succeeds; EOF if an error is generated.

See also

[gets\(\)](#), [putc\(\)](#), [putchar\(\)](#)

qsort

Perform a quick sort.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>qsort.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
void qsort(
void *base,           Pointer to storage destination of array to be sorted
size_t n,             Number of elements
size_t w,             Size of on element
int (*fcmp)())        Pointer to address of comparison function
```

Explanation

Quick-sorts a table of n items (of item size w) starting with *base*, with *fcmp* as the comparison function.

rand

Generate a random number.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>rand.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

int rand(*void*)

Explanation

Generates a pseudo-random number from 0 to RAND_MAX (0x7FFF=32767).

Return value

The generated pseudo-random number.

See also

[srand\(\)](#)

realloc

Change heap memory allocations.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>malloc.h</i>	2.x	12/14/98

Syntax

```
void *realloc(
void *block,           Pointer to a block allocated by a function such as malloc()
size_t s;              New size
```

Explanation

Takes a previously allocated *block* and contracts it or expands it to *s* bytes. If *block* is NULL, this function works in the same way as malloc.

Return value

The address of the reallocated block. May be different from the old address.

If the allocation fails, the function returns NULL, and the old block is not released.

See also

[calloc\(\)](#), [malloc\(\)](#), [free\(\)](#)

setjmp

Defines non-local jump destination.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>setjmp.h</i>	2.x	12/14/98

Syntax

```
int setjmp(  
jmp_buf p)           Environment storage variable
```

Explanation

Stores the destination information for a non-local jump at p. If longjmp(p, val) is executed, the system returns from setjmp().

Return value

Returns the value given to the second argument of longjmp() when the jump is executed.

See also

[longjmp\(\)](#)

srand

Initialize the random number generator.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>rand.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
void srand(  
u_long seed)           Random number seed
```

Explanation

Sets a new starting point for random number generation. The default is 1.

See also

[rand\(\)](#)

strcat

Concatenate character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strcat(  
char *dest,           Pointer to destination string  
char *src)            Pointer to source string
```

Explanation

Appends the character string *src* to the end of the character string *dest*.

Return value

Address of destination string (*dest*).

See also

[strncat\(\)](#)

strchr

Search for the first location at which a character appears in a string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strchr(  
char *s,           Pointer to character string searched  
char c)            Character searched for
```

Explanation

Searches for the first location at which the character *c* appears in the character string *s*.

Return value

Address of the location at which *c* appears. If *c* has not been found, NULL is returned.

See also

[strchr\(\)](#), [strpbrk\(\)](#)

strcmp

Compare character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
int strcmp(  
char *s1,           Pointer to character string 1  
char *s2)           Pointer to character string 2
```

Explanation

Compares the character string *s2* with the character string *s1*, treating each character as an unsigned char.

Return value

<0 if *s1* < *s2*

0 if *s1* = *s2*

>0 if *s1* > *s2*

See also

[strncmp\(\)](#)

strcpy

Copy a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

char *strcpy(

char **dest*, Pointer to destination location.

char **src*) Pointer to source character string

Explanation

Copies the character string *src* to the character string *dest*.

Return value

Pointer to destination string (*dest*).

See also

[strncpy\(\)](#)

strcspn

Search for a string of characters not included in the a character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
int strcspn(
char *s1,           Pointer to string
char *s2)           Pointer to character set
```

Explanation

Returns the length of the first part of the character string *s1* consisting only of characters not included in the character string *s2*.

Return value

The length of the partial character string found.

See also

[strpbrk\(\)](#), [strtok\(\)](#), [strspn\(\)](#)

strlen

Count characters in a string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
int strlen(  
char *s)           Pointer to character string
```

Explanation

Counts the number of characters in string *s*.

Return value

The number of characters in the string.

strncat

Concatenate character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strncat(  
  char *dest,      Pointer to destination string  
  char *src,       Pointer to source string  
  int n)           Number of characters concatenated
```

Explanation

Appends the first *n* characters from *src* to the end of the character string *dest*.

Return value

Pointer to destination string (*dest*).

See also

[strcat\(\)](#)

strncmp

Compare character strings.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
int strncmp(  
  char *s1,           Pointer to character string 1  
  char *s2,           Pointer to character string 2  
  int n)              Number of characters compared
```

Explanation

Compares the first *n* characters of *s1* and *s2*, treating each character as an unsigned char.

Return value

One of the following values, depending on the comparison result (the values are the same as for `strcmp`).

<0 if *s1* < *s2*

0 if *s1* = *s2*

>0 if *s1* > *s2*

See also

[strcmp\(\)](#)

strncpy

Copy a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strncpy(  
  char *dest,           Pointer to destination string  
  char *src,            Pointer to source string  
  int n)               Number of bytes to copy
```

Explanation

Copies the first *n* bytes of *src* to the character string *dest*.

Return value

Pointer to destination string (*dest*).

See also

[strcpy\(\)](#)

strpbrk

Search for the first occurrence of a character in a character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strpbrk(  
  char *s1,           Pointer to character string searched  
  char *s2)           Pointer to character group
```

Explanation

Searches for the first location at which any of the characters contained in the character string s2 appear within the character string s1.

Return value

The address of the first character found. If no character was found, NULL is returned.

See also

[strcspn\(\)](#), [strtok\(\)](#)

strrchr

Search for the last location of a character in a character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strrchr(  
  char *s,           Pointer to character string searched  
  char c)            Character searched for
```

Explanation

Searches for the last occurrence of the character *c* within the character string *s*.

Return value

The address of the last occurrence of *c*. If *c* does not occur, NULL is returned.

See also

[strchr\(\)](#), [strpbrk\(\)](#)

strspn

Search for the part of a string consisting solely of characters in a character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
int strspn(  
char *s1,           Pointer to string  
char *s2)           Pointer to character set
```

Explanation

Returns the length of the first part of the character string *s1* which consists solely of characters included in the character string *s2*.

Return value

The length of the partial character string found.

See also

[strcspn\(\)](#), [strpbrk\(\)](#)

strstr

Search for the location of a partial character string.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

```
char *strstr(  
  char *s1,           Pointer to character string searched  
  char *s2)           Pointer to string searched for
```

Explanation

Searches for the first location of character string s2 within character string s1.

Return value

The address of s2. If it was not found, the function returns NULL.

See also

[strchr\(\)](#)

strtok

Search for a string demarcated by characters in a character set.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>strings.h</i>	2.x	12/14/98

Syntax

**char *strtok(
char *s1,
char *s2)** Pointer to character string searched
 Pointer to separator characters

Explanation

Treats character string *s1* as a set of tokens punctuated by one or more characters from the separator character string *s2*. The first call in the sequence searches *s1* for the first character that is not contained within *s2*.

The first time `strtok()` is called, the starting address of the first token of *s1* is returned, and a NULL character is written in immediately after this token. The address of *s1* is stored in the function, and then, when `strtok()` is called with NULL entered as the first argument, a search is carried out until there are no tokens left in the character string *s1*.

Return value

The starting address of the tokens found in *s1*. If it does not find any *s1* tokens, `strtok()` returns NULL.

See also

[strcspn\(\)](#), [strpbrk\(\)](#)

strtol

Convert a character string to a long.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>convert.h</i>	2.x	12/14/98

Syntax

```
long strtol(
char *s,           Pointer to character string
char **endp,       Storage destination of pointer to a non-convertible character string
unsigned int base) Radix specification
```

Explanation

Converts a character string *s* to a long (the same as an int in R3000). *s* must be formatted as follows:

```
[ws][sn][ddd]
[ws]           white space (may be omitted)
[sn]          sign (may be omitted)
[ddd]         number string (may be omitted)
```

The value of *base* determines the format of *[ddd]*. The letters a (or A) thru z (or Z) are ascribed values from 10-35. Only values less than *base* may be included in *[ddd]*. For some values of *base*, optional characters may precede the sequence of letters and digits following the sign (if present).

Table 2-2

Base Value	Optional Characters
2	0b, 0B
8	"O," "o"
16	0x, 0X

The function `strtol()` stops converting when it encounters a non-convertible character, and if *endp* is not NULL, it sets *endp* as the pointer to the character at which it stopped converting.

Return value

The result obtained by converting the input value *s* to a long. If an error is generated, it returns 0.

See also

[atol\(\)](#), [strtoul\(\)](#)

strtoul

Convert a character string to an unsigned long.

Library	Header File	Introduced	Documentation Date
libc\libc2.lib	convert.h	2.x	12/14/98

Syntax

u_long strtoul(
char *s, Pointer to character string
char **endp, Storage destination of pointer to a non-convertible character string
int base) Radix specification

Explanation

Converts a character string s to unsigned long type (the same as unsigned int type in R3000). s must be formatted as follows.

[ws][sn][ddd]
[ws] white space (may be omitted)
[sn] sign (may be omitted)
[ddd] number string (may be omitted)

The value of base determines the format of [ddd]. The letters a (or A) thru z (or Z) are ascribed values from 10-35. Only values less than base may be included in [ddd]. For some values of base, optional characters may precede the sequence of letters and digits following the sign (if present).

Table 2-3

Base Value	Optional Characters
2	0b, 0B
8	"O," "o"
16	0x, 0X

The function strtoul() stops converting when it encounters a non-convertible character, and if endp is not NULL, it sets endp as the pointer to the character at which it stopped converting.

Return value

The result obtained by converting the input value s to a long.

See also

atol(), strtol()

toascii

Mask bit 7 of the input value.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>ctype.h</i>	2.x	12/14/98

Syntax

`toascii (c)` Value

Explanation

This macro returns an ASCII value equal to the low 7 bits of the input.

Return value

The low 7 bits of the input value *c*.

See also

`isXXXX()`

tolower

Convert a letter to lower-case.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>ctype.h</i>	2.x	12/14/98

Syntax

tolower(c) Character

Explanation

This macro converts a character *c* to lower case. The behavior of this macro when it is given a value not an upper-case letter is undefined.

Return value

The lower-case letter that corresponds to *c*.

See also

[toupper\(\)](#), [isXXXX\(\)](#)

toupper

Converts a character to upper case.

Library	Header File	Introduced	Documentation Date
<i>libc\libc2.lib</i>	<i>ctype.h</i>	2.x	02/15/98

Syntax

toupper(c) Character

Explanation

This macro converts a character *c* to upper case. The behavior of this macro when it is given a value not a lower-case letter is undefined.

Return value

The upper-case letter that corresponds to the character *c*.

See also

[tolower\(\)](#), [isXXXX\(\)](#)

Chapter 3: Math Library

Table of Contents

Functions

acos	3-3
asin	3-4
atan	3-5
atan2	3-6
atof	3-7
ceil	3-8
cos	3-9
cosh	3-10
exp	3-11
fabs	3-12
floor	3-13
fmod	3-14
frexp	3-15
hypot	3-16
ldexp	3-17
log	3-18
log10	3-19
modf	3-20
pow	3-21
printf2	3-22
sin	3-23
sinh	3-24
sprintf2	3-25
sqrt	3-26
strtod	3-27
tan	3-28
tanh	3-29

acos

Arccosine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double **acos**(
double *x*) Value whose arccosine is to be determined, ranging from -1 to 1

Explanation

Determines the arccosine of *x*.

Return value

Arccosine of *x*, ranging from 0 to pi.

Error handling: if `fabs(x)>1`, 0 is returned, and `math_errno` is set to EDOM (domain error).

See also

[cos\(\)](#), [asin\(\)](#), [atan\(\)](#), [atan2\(\)](#)

asin

Arcsine.

Library	Header File	Introduced	Documentation Date
libmath.lib	libmath.h	3.0	12/14/98

Syntax

**double asin(
double x)** Value whose arcsine is to be determined, ranging from -1 to 1.

Explanation

Determines the arcsine of x.

Return value

Arcsine of x, ranging from $-\pi/2$ to $\pi/2$.

Error handling: if $\text{fabs}(x) > 1$, 0 is returned, and `math_errno` is set to EDOM (domain error).

See also

[sin\(\)](#), [acos\(\)](#), [atan\(\)](#), [atan2\(\)](#)

atan

Arctangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double atan(
double x)** Value whose arctangent is to be calculated

Explanation

Determines the arctangent of x.

Return value

Arctangent of x, ranging from -pi/2 to pi/2 radians.

See also

[tan\(\)](#), [asin\(\)](#), [acos\(\)](#), [atan2\(\)](#)

atan2

Arctangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double atan2(
double x, double y)** Floating-point values

Explanation

Determines the arctangent of x/y . If x and y are 0, a value of 0 is returned.

Return value

Arctangent of x/y , ranging from $-\pi$ to π .

See also

[acos\(\)](#), [asin\(\)](#), [tan\(\)](#), [atan\(\)](#)

atof

Convert a string to a floating-point equivalent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double atof(
char *s) Pointer to a string

Explanation

Converts a string "s" to its floating-point (double type) equivalent.

Return value

The result from converting input string "s" to a double floating point equivalent.

Error handling: if there is an overflow error, either +HUGE_VAL(1.797693134862316e+308) or -HUGE_VAL depending on the sign, is returned, and math_erno is set to ERANGE (range error). If there is an underflow, 0 is returned, and math_erno is set to ERANGE (range error).

See also

[strtod\(\)](#)

ceil

Minimum integer not less than x (ceiling function).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double **ceil**(
double x) Floating-point value

Explanation

Determines the minimum integer (double type) not less than x .

Return value

Minimum integer (double type) not less than x .

See also

[floor\(\)](#)

COS

Cosine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double cos(
double *x*)** Angle in radians

Explanation

Determines the cosine of *x*.

Return value

Cosine of *x* (cos(*x*)).

See also

[sin\(\)](#), [tan\(\)](#), [acos\(\)](#)

cosh

Hyperbolic cosine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double cosh(
double x) Angle in radians

Explanation

Determines the hyperbolic cosine of x .

Return value

Hyperbolic cosine of x ($\cosh(x)$).

See also

[sinh\(\)](#), [tanh\(\)](#)

exp

Exponent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double exp(
double x)** Floating-point value

Explanation

Determines the exponential of x.

Return value

e raised to the x-th power (e^x).

See also

[pow\(\)](#), [log\(\)](#), [ldexp\(\)](#)

fabs

Absolute value (macro).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double fabs(
double *x*)** Floating-point value

Explanation

This macro determines the absolute value of a double.

Return value

Absolute value of *x*.

See also

abs (see libc), labs() (see libc)

floor

Maximum integer not more than x (lower function).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double floor(
double x)** Floating-point value

Explanation

Determines the maximum integer (double type) not more than x.

Return value

Maximum integer not more than x (double type)

See also

[ceil\(\)](#)

fmod

Floating-point remainder resulting from x/y .

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double fmod(
double x, double y)** Floating-point values

Explanation

Determines the floating-point remainder resulting from x/y . The sign of the return value is the same as that of x .

Return value

Floating-point remainder resulting from x/y . If y is 0, 0 is returned.

See also

[modf\(\)](#)

frexp

Resolve into a normalized fraction and a power of 2.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double frexp(

double *x*,

int **n*)

Floating-point value

Pointer to the part that is a power of 2

Explanation

Resolves *x* into a fraction in the interval $[1/2, 1)$ (that is, $1/2 \leq x < 1$), and a power of 2. The fractional part is returned, and the power of 2 is stored in *n*.

A pair of square brackets `[]` indicates a closed area, while a pair of parentheses `()` indicates an open area.

Return value

The normalized fraction.

hypot

Absolute value of a complex number.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double hypot(
double x, double y)** Floating-point values

Explanation

Computes the square root of the sum of the squares of *x* and *y*.

Return value

Square root of the sum of (*x***2) and (*y***2).

ldexp

Calculate a real number from a mantissa and an exponent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double ldexp(

double *x*, Floating-point value
int *n*) Integral exponent

Explanation

Determines a real number from a mantissa and an exponent.

Return value

Value of *x* multiplied by 2 raised to the *n*th power.

log

Natural logarithm.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double log(

double x) Value subjected to logarithmic operation

Explanation

Determines the natural logarithm of x.

Return value

Logarithm of x ($\ln(x)$) for $x > 0$.

Error handling: If $x = 0$, 1 is returned, and `math_errno = ERANGE` (range error). If $x < 0$, 0 is returned, and `math_errno = EDOM` (domain error).

See also

[exp\(\)](#), [log10\(\)](#)

log10

Base 10 logarithm.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double log10(

double *x*) Value subjected to logarithmic operation

Explanation

Determines the logarithm of *x* whose base is 10.

Return value

Logarithm of *x* whose base is 10 (log₁₀(*x*))

x must be greater than zero. Otherwise, an error results: If *x* = 0, 1 is returned, and math_errno = ERANGE (range error). If *x* < 0, 0 is returned, and math_errno = EDOM (domain error).

See also

[log\(\)](#)

modf

Separate a double into integral and fractional parts.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double modf(

double x,

double *y)

Floating-point value

Pointer to the integral part

Explanation

Separates x into integral and fractional parts. The integral part is stored in y, and the return value is the fractional part. The signs of both parts are the same as the sign of x.

Return value

Fractional part of x.

See also

[fmod\(\)](#)

pow

Raise a double to a power.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

```
double pow(
double x,           Floating-point value
double y)           Pointer to the integral part
```

Explanation

Raises x to the y -th power.

Return value

x raised to the y -th power (x^{**y}).

Table 3-1

Condition	Return value	Error (math_errno)
$x==0 \ \&\& \ y>0$	0	Domain error (EDOM)
$x==0 \ \&\& \ y<=0$	1	Domain error (EDOM)
$x<0 \ \&\& \ [y \text{ not an integer}]$	0	Domain error (EDOM)

See also

[exp\(\)](#), [sqrt\(\)](#)

printf2

Convert formatted output as standard output (with floating-point and double-precision arguments).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	2/24/99

Syntax

```
int printf2(
  const char *fmt[,argument...])
```

Pointer to input format character string

Explanation

The conversion directives [f] [e] [E] [g] and [G] can be used.

Stack consumption is greater than with printf().

Note: When printf() (or printf2(), putchar(), or puts()) is being executed in the main flow, and an interrupt occurs, text corruption or a hang-up can result if printf() is called during the interrupt. Therefore, pay attention to the call timing when calling printf() in an interrupt.

Return value

Output character length.

See also

[sprintf2\(\)](#)

sin

Sine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double sin(
double *x*)** Angle in radians

Explanation

Determines the sine of *x*.

Return value

Sine of *x* (sin(*x*)).

See also

[cos\(\)](#), [tan\(\)](#), [asin\(\)](#)

sinh

Hyperbolic sine.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double sinh(
double x)** Angle in radians

Explanation

Determines the hyperbolic sine of x.

Return value

Hyperbolic sine of x (sinh(x)).

See also

[cosh\(\)](#), [tanh\(\)](#)

sprintf2

Format output to a string (with floating-point and double-precision arguments).

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

int sprintf2(
char *s,
const char *fmt[,argument...])

Pointer to destination string

Pointer to input format character string

Explanation

The conversion directives [f] [e] [E] [g] and [G] can be used.

Stack consumption is greater than with sprintf.

Return value

Output character length.

See also

[printf2\(\)](#)

sqrt

Square root.

12/14/98	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double sqrt(
double x)** Non-negative floating-point value

Explanation

Determines the non-negative square root of x .

Error processing: if $x < 0$, zero is returned, and `math_errno = EDOM` (domain error).

Return value

Square root of x .

See also

[pow\(\)](#)

strtod

Convert a string to a floating-point equivalent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

double strtod(

char *s, Input string

char **endp) Pointer to a string that was unable to be converted (output)

Explanation

Converts a string to a double type floating-point equivalent.

s must be one of the following:

[ws][sn][ddd]

[ws] White space (may be omitted)

[sn] Sign (may be omitted)

[ddd] Number string (may be omitted)

Stops converting upon encountering a character that was unable to be converted. If *endp* is not NULL, the pointer to the character in error is set to *endp*.

Return value

The result from converting s to a floating point double type.

Error handling: if the converted value overflows, either +HUGE_VAL(1.797693134862316e+308) or -HUGE_VAL according to the sign, is returned. 0 is returned for an underflow case, or if no conversion could be performed. In either case, math_errno = ERANGE (range error).

See also

[atof\(\)](#)

tan

Tangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double tan(
double x)** Angle in radians

Explanation

Determines the tangent of x.

Return value

Tangent of x (tan(x)).

See also

[sin\(\)](#), [cos\(\)](#), [atan\(\)](#), [atan2\(\)](#)

tanh

Hyperbolic tangent.

Library	Header File	Introduced	Documentation Date
<i>libmath.lib</i>	<i>libmath.h</i>	3.0	12/14/98

Syntax

**double tanh(
double *x*)** Angle in radians

Explanation

Determines the hyperbolic tangent of *x*.

Return value

Hyperbolic tangent of *x* (tanh(*x*)).

See also

[sinh\(\)](#), [cosh\(\)](#)

Chapter 4: Memory Card Library

Table of Contents

Functions

InitCARD	4-3
StartCARD	4-4
StopCARD	4-5
_bu_init	4-6
_card_auto	4-7
_card_chan	4-8
_card_clear	4-9
_card_format	4-10
_card_info	4-11
_card_load	4-12
_card_read	4-13
_card_status	4-14
_card_wait	4-15
_card_write	4-16
_new_card	4-17

InitCARD

Initialize Memory Card BIOS.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
void InitCARD(
long val)           Specify sharing with controller
```

Explanation

Initializes the Memory Card BIOS and enters an idle state. *val* specifies whether or not there is sharing with the controller. (0: not shared; 1: shared.)

When the BIOS is subsequently put into operation by StartCARD(), the low-level interface functions that begin with “_card” can be used directly.

The Memory Card file system uses these interfaces internally, so InitCARD() needs to be executed before _bu_init().

There is no effect on the controller.

See also

[_bu_init\(\)](#)

StartCARD

Start Memory Card BIOS.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

void StartCARD(void)

Explanation

Changes the Memory Card BIOS initialized by InitCARD() to a run state.

Performs ChangeClearPAD(1) internally.

See also

[InitCARD\(\)](#), [StopCARD\(\)](#), [_bu_init\(\)](#), [ChangeClearPAD\(\)](#) (see libapi)

StopCARD

Stop Memory Card BIOS.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

void StopCARD(void)

Explanation

Changes Memory Card BIOS to an idle state--the same state as that immediately after executing InitCARD().

It also stops the controller. It is necessary to call StartPAD() to start the controller.

See also

[InitCARD\(\)](#), [StartCARD\(\)](#), [_bu_init\(\)](#), [ChangeClearPAD\(\)](#) (see libapi)

_bu_init

Initialize Memory Card file system.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

void_bu_init(void)

Explanation

Initializes the Memory Card file system. This file system is not initialized automatically, so it is necessary to call this function.

See also

[InitCARD\(\)](#), [StartCARD\(\)](#), [StopCARD\(\)](#)

_card_auto

Set automatic format function.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

**long _card_auto(
long *val*)** Indicates automatic formatting

Explanation

When *val* is 0, the automatic format function is disabled; when *val* is 1, it is enabled.

This function should be used for testing purposes only.

Return value

Previously set automatic format value.

_card_chan

Get a Memory Card BIOS event.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

long _card_chan(void)

Explanation

Returns the device number of the Memory Card that just generated an event.

Return value

2-digit hex device number.

See also

[card_status\(\)](#), [_card_wait\(\)](#)

`_card_clear`

Clear unconfirmed flags.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
long _card_clear(  
long chan)
```

Port number x 16 + Card number

Explanation

Performs a dummy write to the system management area of the card and clears the card's unconfirmed flags.

When calculating *chan*, "port number" is 0 for Port 1 and 1 for Port 2. "Card number" is zero when a standard controller is connected, and may be in the range 0-3 if a Multi Tap is connected.

This function executes asynchronously, so it returns immediately. Processing completion is communicated by an event. (See table below.) In order to use this command with multiple slots in a Multi Tap, you must wait until processing has completed before sending another `_card_clear()` call.

Table 4-1: Events on completion of processing

Source Descriptor/Event Class	Contents
HwCARD/EvSpIOE	Ends process
HwCARD/EvSpTIMOUT	Card not connected
HwCARD/EvSpNEW	New card detected
HwCARD/EvSpERROR	Error generated
HwCARD/EvSpUNKOWN	Source unknown

Return value

1 if registration successful, otherwise 0.

See also

[card_info\(\)](#)

_card_format

Format the Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

long _card_format(
long *chan*) Port number x 16 + Card number

Explanation

Formats the Memory Card. When calculating *chan*, “port number” is 0 for Port 1 and 1 for Port 2. “Card number” is zero when a standard controller is connected, and may be in the range 0-3 if a Multi Tap is connected.

Does not enter critical section. Synchronous functions are blocked for approximately 144 Vsync.

Return value

1 if formatting is successful, otherwise 0.

See also

[_card_load\(\)](#)

_card_info

Get card status.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

**long _card_info(
long *chan*)** Port number x 16 + Card number

Explanation

Tests the connection of the Memory Card specified in *chan*.

When calculating *chan*, “port number” is 0 for Port 1 and 1 for Port 2. “Card number” is zero when a standard controller is connected, and may be in the range 0-3 if a Multi Tap is connected.

This function executes asynchronously, so it returns immediately. Processing completion is communicated by an event. (See table below.) In order to use this command with multiple slots in a Multi Tap, you must wait until processing has completed before sending another `_card_info()` call.

Table 4-2: Posts an event on completion of processing

Source Descriptor/Event Class	Description
SwCARD/EvSpIOE	Connected
SwCARD/EvSpTIMOUT	Not connected
SwCARD/EvSpNEW	No writing after connection
SwCARD/EvSpERROR	Generates an error

Do not use `_new_card()` to suppress EvSpNEW.

Return value

1 if registration successful, otherwise 0.

See also

[_card_clear\(\)](#), [_card_status\(\)](#), [_new_card\(\)](#)

_card_load

Test logical format

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
long _card_load(
long chan)          Port number x 16 + Card number
```

Explanation

Reads file management information for the card specified by *chan* in the file system in order to get asynchronous access using the I/O management service.

When calculating *chan*, “port number” is 0 for Port 1 and 1 for Port 2. “Card number” is zero when a standard controller is connected, and may be in the range 0-3 if a Multi Tap is connected.

`_card_load()` must be called at least once before you can use `open()` on a Memory Card file in `O_NOWAIT` mode. It does not have to be called again unless a card is changed.

This function executes asynchronously, so it returns immediately. Processing completion is communicated by an event. (See table below.) In order to use this command with multiple slots in a Multi Tap, you must wait until processing has completed before sending another `_card_load()` call.

Table 4-3: Posts an event on completion of processing

Source Descriptor/ Event Class	Contents
SwCARD/EvSpIOE	Read completed
SwCARD/EvSpTIMOUT	Not connected
SwCARD/EvSpNEW	Uninitialized card
SwCARD/EvSpERROR	Generates an error

Return value

1 if the read is successful, otherwise 0.

See also

[format\(\)](#) (see [libcd](#)), [card_info\(\)](#)

`_card_read`

Read one block from the Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```
long _card_read(
long chan,           Port number x 16 + card number
long block,          Target block number
long *buf)            Pointer to 128 byte data buffer
```

Explanation

Reads 128 bytes of buffer data into *buf* from the target block number (*block*) of the Memory Card of the specified channel (*chan*).

When calculating *chan*, “port number” is 0 for Port 1 and 1 for Port 2. “Card number” is zero when a standard controller is connected, and may be in the range 0-3 if a Multi Tap is connected.

This function executes asynchronously so it returns immediately after completion. Actual processing termination is communicated by an event. (See table below.) Multiplex processing to the same card slot can't be performed.

Table 4-4: Events on completion of processing

Source Descriptor / Event Class	Contents
HwCARD/EvSpIOE	Ends processing
HwCARD/EvSpTIMOUT	Card not connected
HwCARD/EvSpNEW	New card detected
HwCARD/EvSpERROR	Error generated
HwCARD/EvSpUNKOWN	Source unknown

This function exists within the low-level interface and is one of the special functions used for testing.

Return value

1 if successful processing registration, otherwise 0.

See also

[_card_write\(\)](#), [open\(\)](#) (see [libapi](#)), [read\(\)](#) (see [libapi](#))

_card_status

Get Memory Card BIOS status.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

long _card_status(
long *drv*) Port number

Explanation

Gets the Memory Card BIOS status of each slot, *drv*. Specify *drv* as 0 for Port 1, 1 for Port 2.
This is a synchronous function.

Return value

If the Memory Card BIOS is in run state, it can return any of the following values.

Table 4–5

Value	State
0x01	Idle processing
0x02	READ processing
0x04	WRITE processing
0x08	Connection test processing registration
0x11	No registered processing (just prior to EvSpTIMOUT generation)
0x21	No registered processing (just prior to EvSpERROR generation)

See also

[card_wait\(\)](#), [_card_chan\(\)](#), [_card_info\(\)](#)

_card_wait

Wait for Memory Card BIOS completion.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

**long _card_wait(
long *drv*)** Sets slot number

Explanation

Wait until registration processing completes for the *drv* slot. Specify *drv* as 0 for Port 1, 1 for Port 2.

Return value

Always 1.

See also

[_card_status\(\)](#), [_card_chan\(\)](#)

_card_write

Write to one block of the Memory Card (for testing only)

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

```

long _card_write(
long chan,           Port number x 16 + card number
long block,          Target block number
long *buf)           Pointer to 128-byte data buffer

```

Explanation

Writes 128 bytes of buffer data pointed to by *buf* to the target block number (*block*) of the Memory Card of the specified channel (*chan*).

When calculating *chan*, “port number” is 0 for Port 1 and 1 for Port 2. “Card number” is zero when a standard controller is connected, and may be in the range 0-3 if a Multi Tap is connected.

This function executes asynchronously, so it returns immediately. Actual processing termination is communicated by an event. (See table below.) Multiplex processing to the same card slot can't be performed; that is, multiple `_card_write()` calls to the same Multi Tap cannot be processed synchronously.

Table 4-6: Events on completion of processing

Source Descriptor/Event Class	Contents
HwCARD/EvSpiOE	Ends process
HwCARD/EvSpTIMOUT	Card not connected
HwCARD/EvSpNEW	New card detected
HwCARD/EvSpERROR	Error generated
HwCARD/EvSpUNKOWN	Source unknown

This is a low-level function that should be used for testing only. It bypasses the memory card file system; therefore, in a released product, use the C file-handling routines such as `write()`.

Return value

1 if registration successful, otherwise 0.

See also

[_card_read\(\)](#), [open\(\)](#) (see [libapi](#)), [write\(\)](#) (see [libapi](#))

`_new_card`

Change settings of unconfirmed flag test.

Library	Header File	Introduced	Documentation Date
<i>libcard.lib</i>	<i>libapi.h</i>	3.0	12/14/98

Syntax

`void _new_card(void)`

Explanation

Masks the generation of an EvSpNEW event immediately after `_card_read()` or `_card_write()`.

Terminates immediately even though it is a synchronous function.

See also

[_card_clear\(\)](#), [_card_read\(\)](#), [_card_write\(\)](#)

Chapter 5: Extended Memory Card Library

Table of Contents

Functions

MemCardAccept	5-3
MemCardCallback	5-4
MemCardClose	5-5
MemCardCreateFile	5-6
MemCardDeleteFile	5-7
MemCardEnd	5-8
MemCardExist	5-9
MemCardFormat	5-10
MemCardGetDirent	5-11
MemCardInit	5-12
MemCardOpen	5-13
MemCardReadData	5-14
MemCardReadFile	5-15
MemCardStart	5-16
MemCardStop	5-17
MemCardSync	5-18
MemCardUnformat	5-19
MemCardWriteData	5-20
MemCardWriteFile	5-21

MemCardAccept

Check Memory Card status.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

```
long MemCardAccept(
long chan)           port number + card number
                    port number(port A: 0x00; port B: 0x10)
                    card number (normally 0)
```

Explanation

Tests connection with the Memory Card specified by *chan*. If the card is connected, additional information is obtained. If the card is new, `_card_clear()` and `_card_load()` are executed, allowing the use of file access functions.

`MemCardAccept()` must be executed before using file access functions such as `MemCardOpen()`. `MemCardAccept()` does not need to be called again as long as the card is not swapped.

The function is asynchronous and returns immediately. (Multiple instances can't be registered.) Use `MemCardSync()` or an exit callback to determine completion and get the result, which is one of the following:

Table 5-1

Value	Macro	Status
0x00	McErrNone	Connected
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Bad card
0x03	McErrNewCard	New card (card was replaced)
0x04	McErrNotFormat	Not formatted

The maximum time required to perform this operation is 76 VSyncs. Approximately 4 VSyncs are needed if a card is not present.

A new card is detected only once and returns `McErrNewCard`. Subsequent calls return `McErrNone`.

Return value

1 if registration successful, otherwise 0.

See also

[MemCardOpen\(\)](#), [MemCardReadFile\(\)](#), [MemCardWriteFile\(\)](#), [MemCardExist\(\)](#)

MemCardCallback

Define exit callback.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

```
MemCB MemCardCallback(
MemCB func)
```

pointer to callback function

Explanation

Sets the callback function (*func*) to be triggered when an asynchronous function completes. If *func* is 0, no callback is generated.

The following format is used for exit callback functions:

```
typedef void (*MemCB)( unsigned long cmds, unsigned long result )
```

cmds: the completed asynchronous function (see below)

result: the execution result from the asynchronous function

Allowed value for *cmds*:

Table 5-2

Value	Macro	Function
0x01	McFuncExist	MemCardExist
0x02	McFuncAccept	MemCardAccept
0x03	McFuncReadFile	MemCardReadFile
0x04	McFuncWriteFile	MemCardWriteFile
0x05	McFuncReadData	MemCardReadData
0x06	McFuncWriteData	MemCardWriteData

See the sections on the respective functions for details of the result value.

Return value

The address of the previously set callback.

See also

MemCardAccept(), MemCardExist(), MemCardReadFile(), MemCardWriteFile(), MemCardReadData(), MemCardWriteData(), MemCardSync()

MemCardClose

Close file.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

void MemCardClose(void)

Explanation

Closes the file that was opened with MemCardOpen(). It is an asynchronous function that exits immediately.

See also

[MemCardOpen\(\)](#)

MemCardCreateFile

Create a new file in the Memory Card

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

long MemCardCreateFile(

long *chan*,

port number + card number

port number (port A: 0x00, port B: 0x10)

card number (normally 0)

char **file*,

filename

long *blocks*)

number of blocks

Explanation

Creates the specified file in the Memory Card. It is a synchronous function; blocking time is 1 - 4 VSyncs for normal exit, 4 - 76 VSyncs otherwise. It doesn't enter a critical section.

The *block* parameter is given in units of 8192 bytes.

Return value

Table 5-3

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Communication error generated
0x04	McErrNotFormat	Not formatted
0x06	McErrAlreadyExist	File already exists
0x07	McErrBlockFull	Not enough available blocks
-1	None	A non-synchronous function is active.

See also

[MemCardOpen\(\)](#), [MemCardWriteFile\(\)](#), [MemCardDeleteFile\(\)](#)

MemCardDeleteFile

Delete file from Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	2/24/99

Syntax

long MemCardDeleteFile(

long *chan*, port number + card number
 port number (port A: 0x00, port B: 0x10)
 card number (normally 0)
char **file*) filename

Explanation

Deletes the specified file from the Memory Card. It is a synchronous function; blocking time: 1 - 4 VSyncs for normal exit. 4 - 76 VSyncs otherwise. Does not enter critical section.

Return value

Table 5-4

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Communication error generated (*1)
0x04	McErrNotFormat	Not formatted
0x05	McErrFileNotExist	File not found
-1	None	A non-synchronous function is active.

(*1: The same error code is also returned if the deleted file was an active PDA application. Because of the PDA kernel locking mechanism, the active PDA application cannot be deleted.

See also

[MemCardCreateFile\(\)](#)

MemCardEnd

Terminate Memory Card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

void MemCardEnd(void)

Explanation

Terminates the Memory Card system. It is a synchronous function.

MemCardStop() needs to be executed first if the system was activated from MemCardStart().

See also

[MemCardInit\(\)](#), [MemCardStart\(\)](#), [MemCardStop\(\)](#)

MemCardExist

Get connection status of card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

```
long MemCardExist(
long chan)           port number + card number
                    port number (port A: 0x00, port B: 0x10)
                    card number (normally 0)
```

Explanation

Tests the connection status of the Memory Card specified by *chan*. `MemCardExist()` is faster than `MemCardAccept()`, since it checks only the presence of the card. `MemCardAccept()` must be used for more detailed information, such as whether the card is formatted. If cards are swapped, `MemCardAccept()` must be executed before using file access functions such as `MemCardOpen()`.

The function is asynchronous and exits immediately. Multiple instances of the function cannot be registered. Use `MemCardSync()` or an exit callback to determine completion and obtain the result of the operation, as shown below:

Table 5-5

Value	Macro	Status
0x00	<code>McErrNone</code>	Connected
0x01	<code>McErrCardNotExist</code>	Not connected
0x02	<code>McErrCardInvalid</code>	Bad card
0x03	<code>McErrNewCard</code>	New card (card was replaced)

The time required is approximately 4 VSyncs.

When a new card is detected (`McErrNewCard`), you must call `MemCardAccept()` to clear the new card status, before performing any other operations.

Return value

1, if the command was successfully registered; 0 otherwise.

See also

[MemCardAccept\(\)](#)

MemCardFormat

Format Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

**long MemCardFormat(
long *chan*)**

port number + card number
port number (port A: 0x00, port B: 0x10)
card number (normally 0)

Explanation

Formats the specified Memory Card. Synchronous function. Blocking time: approx. 144 VSyncs. Does not enter critical section.

Return value

Table 5-6

Value	Macro	Status
0x00	McErrNone	Connected
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
-1	None	A non-synchronous function is active

See also

[MemCardUnformat\(\)](#)

MemCardGetDirentry

Get directory information from Memory Card.

Library	Header File	Introduced	Documentation Date
libmcrd.lib	libmcrd.h	4.0	2/24/99

Syntax

```
long MemCardGetDirentry(
    long chan,                port number + card number
                              port number (port A: 0x00, port B: 0x10)
                              card number (normally 0)
    char *name,               filename to be searched (wildcards can be used)
    struct DIRENTRY *dir,     pointer to structure to hold information about matching files
    long *files,              pointer to buffer to hold number of matching files
    long ofs,                 offset for entry. Specifies the number of files to skip from the first file that
                              matches before saving to the buffer (0 - 14).
    long max)                 maximum number of entries to store in the buffer
```

Explanation

Finds files matching the filename pattern *name*. Data for these files are stored in *dir*, and the total number of matching files is returned in *files*. The buffer must be prepared by the user application.

Synchronous function. Blocking time: 1 - 2 VSyncs for normal exit. Otherwise, 4 - 76 VSyncs.

Wildcard characters can be used in the filename pattern: "?" for any single character; "*" for any number of characters. Characters following * are ignored.

Return value

Table 5-7

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Bad card
-1	None	A non-synchronous function is active.

MemCardInit

Initialize Memory Card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

void MemCardInit(

long val)

Use of control routine in ROM (0: do not use, 1: use)

Explanation

Initializes the Memory Card system. If the system is subsequently activated with MemCardStart(), libmcrd functions (those beginning with "MemCard") are available. MemCardInit() should be executed after InitPAD(), InitGUN(), or InitTAP().

val should be set to 0 when using libtap or libgun.

MemCardInit() requires 60 - 70 VSynchs to complete. It cannot be executed twice.

See also

[MemCardEnd\(\)](#), [MemCardStart\(\)](#), [MemCardStop\(\)](#)

MemCardOpen

Open file.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

```
long MemCardOpen(
long chan,           port number + card number
                    port number (port A: 0x00, port B: 0x10)
                    card number (normally 0)
char *file,          filename
long flag)           specifies method with which to open
                    (read only: O_RDONLY, write only: O_WRONLY)
```

Explanation

Opens the specified Memory Card file with the method specified by *flag*. Once the file is open, `MemCardReadData()` and `MemCardWriteData()` can be used.

Methods cannot be combined (O_RDONLY|O_WRONLY). Multiple files cannot be opened.

Synchronous function. Blocking time: Exits immediately for normal completion. Otherwise, 4 - 76 VSyncs.

Return value

Table 5-8

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Card is not connected
0x02	McErrCardInvalid	Bad card
0x04	McErrNotFormat	Not formatted
0x05	McErrFileNotExist	File not found
-1	None	Either another file is already open or a non-synchronous function is active in the background.

See also

[MemCardReadData\(\)](#), [MemCardWriteData\(\)](#), [MemCardClose\(\)](#)

MemCardReadData

Read data from Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

long MemCardReadData(

unsigned long *adrs, pointer to destination buffer in main memory
long offset, offset in bytes from which to read, where the start of the file is defined to be 0

long bytes) number of bytes to read (multiple of 128)

Explanation

Reads data from the Memory Card file previously opened in MemCardOpen() and stores it in the buffer pointed to by *adrs*.

It is an asynchronous function and exits immediately. Multiple instances cannot be registered.

Use MemCardSync() or an exit callback to determine completion and obtain the result of the operation. (The time required is approximately 1 VSync overhead + approximately 130 VSyncs per block (8192 bytes).

bytes is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

The function result can have the values shown below.

Table 5-9

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)

Return value

1 if operation was registered successfully. Otherwise, 0.

See also

[MemCardOpen\(\)](#), [MemCardSync\(\)](#)

MemCardReadFile

Read file from Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

long MemCardReadFile(

long *chan*,

port number + card number

port number (port A: 0x00, port B: 0x10)

card number (normally 0)

char **file*,

filename

unsigned long **adrs*,

pointer to destination buffer in main memory

long *offset*,

offset in bytes from which to read, where the start of the file is defined to be 0

long *bytes*)

number of bytes to read (multiple of 128)

Explanation

Reads data from the specified Memory Card file and stores it in the buffer pointed to by *adrs*. *bytes* is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

This function is asynchronous and returns immediately. Multiple instances cannot be registered. Use `MemCardSync()` or an exit callback to determine completion and obtain the result of the operation, as shown below.

Table 5-10

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)
0x05	McErrFileNotExist	File cannot be found

Required time: approximately 1 VSync overhead + approximately 130 VSyncs per block (8192 bytes).

`MemCardOpen()` and `MemCardReadData()` are executed within `MemCardReadFile()`. If `MemCardOpen()` is executed on a file which is already open, an error is generated and the value 0 is returned.

Return value

1 if operation was registered successfully. If the file was already open, or another asynchronous function was already registered, 0 is returned.

See also

[MemCardOpen\(\)](#), [MemCardReadFile\(\)](#), [MemCardSync\(\)](#)

MemCardStart

Start Memory Card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

void MemCardStart(void)

Explanation

Places the Memory Card system, previously initialized with MemCardInit() in an active state. Internally, eight events such as HwCARD and SwCARD are opened.

Asynchronous Function. Exits immediately.

See also

[MemCardInit\(\)](#), [MemCardStop\(\)](#)

MemCardStop

Stop Memory Card system.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

void MemCardStop(void)

Explanation

Stops the Memory Card system activated by MemCardStart(). Various events are closed.

Asynchronous Function. Exits immediately.

See also

[MemCardInit\(\)](#), [MemCardStart\(\)](#), [MemCardStop\(\)](#)

MemCardSync

Wait for completion of an asynchronous function or check status.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	12/14/98

Syntax

long MemCardSync(

long mode,

0: wait for termination of asynchronous function

1: check current status and return immediately

long *cmds,

pointer to the terminated asynchronous function

long *result)

pointer to execution results from the asynchronous function

Explanation

If *mode* is 0, this function waits for termination of an asynchronous function such as `MemCardAccept()` and `MemCardReadFile()`. The execution time depends on the corresponding asynchronous function.

If *mode* is 1, it exits immediately and returns the status of the asynchronous function (see Return value).

cmds stores the operation code corresponding to the terminated asynchronous function:

Table 5-11

Value	Macro	Function
0x01	McFuncExist	MemCardExist
0x02	McFuncAccept	MemCardAccept
0x03	McFuncReadFile	MemCardReadFile
0x04	McFuncWriteFile	MemCardWriteFile
0x05	McFuncReadData	MemCardReadData
0x06	McFuncWriteData	MemCardWriteData

Return value

0: Still active

1: Terminated

-1: No registered process

See also

[MemCardAccept\(\)](#), [MemCardExist\(\)](#), [MemCardReadFile\(\)](#), [MemCardWriteFile\(\)](#), [MemCardReadData\(\)](#), [MemCardWriteData\(\)](#), [MemCardCallback\(\)](#)

MemCardUnformat

Uninitialize a Memory Card (for debugging only).

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.3	12/14/98

Syntax

```
long MemCardUnformat(
long chan)                port number + card number
                           port number (port A: 0x00, port B: 0x10)
                           card number (default 0)
```

Explanation

Puts Memory Card in uninitialized (unformatted) state. Synchronous function.

MemCardUnformat() is a debugging function that can be used to create an unformatted card. This function should only be used for testing Memory Card initialization during program debugging. It should not be used in an actual title.

Return value

1: Completed successfully. 0: Error. -1: Could not be executed because of an asynchronous function running in the background.

See also

[MemCardFormat\(\)](#)

MemCardWriteData

Write data to Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	8/9/99

Syntax

long MemCardWriteData(

unsigned long **adrs*,

long *offset*,

long *byte*)

pointer to destination buffer in main memory

offset in bytes from which to write, where the start of the file is defined to be 0

number of bytes to write (multiple of 128)

Explanation

Writes data from the buffer pointed to by *adrs* to the Memory Card file previously opened with MemCardOpen().

MemCardWriteData() is asynchronous and exits immediately. Multiple instances cannot be registered. Required time: Approximately 1 VSync overhead + 130 VSyncs per block (8192 bytes)

Use MemCardSync() or an exit callback to determine completion and obtain the result of the operation.

bytes is specified in units of 128. If a number that is not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

The function result can have the values shown below.

Table 5-12

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)

Return value

1 if the operation was registered successfully, 0 otherwise.

See also

[MemCardOpen\(\)](#), [MemCardSync\(\)](#)

MemCardWriteFile

Write file to Memory Card.

Library	Header File	Introduced	Documentation Date
<i>libmcrd.lib</i>	<i>libmcrd.h</i>	4.0	8/9/99

Syntax

```
long MemCardWriteFile(
long chan,                port number + card number
                           port number (port A: 0x00, port B: 0x10)
                           card number (normally 0)
char *file,               filename
unsigned long *adrs,       pointer to destination buffer in main memory
long offset,              offset in bytes from which to write, where the start of the file is defined to
                           be 0
long bytes)               number of bytes to write (multiple of 128)
```

Explanation

Writes data from the buffer pointed to by *adrs* to the specified Memory Card. If the file is new, it must be created beforehand with `MemCardCreateFile()`. *bytes* is specified in units of 128. If a number not a multiple of 128 is specified, the process is not registered and the operation terminates with a return value of 0.

`MemCardWriteFile()` is an asynchronous function and returns immediately. Multiple instances cannot be registered. Use `MemCardSync()` or an exit callback to determine completion and obtain the result of the operation, as shown below:

The function result can have the values shown below.

Table 5-13

Value	Macro	Status
0x00	McErrNone	Normal exit
0x01	McErrCardNotExist	Not connected
0x02	McErrCardInvalid	Communication error
0x03	McErrNewCard	New card (card swapped)
0x05	McErrFileNotExist	File not found

Required time: Approximately 1 VSync overhead + 130 VSyncs per block (8192 bytes).

`MemCardOpen()` and `MemCardWriteData()` are executed within `MemCardWriteFile()`. If `MemCardOpen()` is executed on a file which is already open, an error is generated and the value 0 is returned.

Return value

1 if operation was registered successfully. If the file was already open, or another asynchronous function was already registered, 0 is returned.

See also

[MemCardCreateFile\(\)](#), [MemCardSync\(\)](#)

Chapter 6: Data Compression Library

Table of Contents

Structures

DECDCCTENV	6-3
ENCSPUENV	6-4

Functions

DecDCTBufSize	6-5
DecDCTGetEnv	6-6
DecDCTIn	6-7
DecDCTInCallback	6-8
DecDCTInSync	6-9
DecDCTOut	6-10
DecDCTOutCallback	6-11
DecDCTOutSync	6-12
DecDCTPutEnv	6-13
DecDCTReset	6-14
DecDCTvlc	6-15
DecDCTvlc2	6-16
DecDCTvlcBuild	6-17
DecDCTvlcSize	6-18
DecDCTvlcSize2	6-19
EncSPU	6-20
EncSPU2	6-22

ENCSPUENV

SPU encode environment attribute structure.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.6	8/27/99

Structure

```
typedef struct {
    short *src;           16-bit PCM data address
    short *dest;          PlayStation® original waveform data
    short *work;          Work area when encode processing
    long size;            16-bit PCM data size(in bytes)
    long loop_start;      PCM data loop start point(in bytes)
    char loop;            Loop waveform generation specification
                        ENCSPU_ENCODE_LOOP: Generate loop waveform data
                        ENCSPU_ENCODE_NO_LOOP: Generate non-loop waveform data
    char byte_swap;       PCM data endian specification
                        ENCSPU_ENCODE_ENDIAN_BIG: 16-bit big endian
                        ENCSPU_ENCODE_ENDIAN_LITTLE: 16-bit little endian
    char proceed;         Whole/Divided encoding specification
                        ENCSPU_ENCODE_WHOLE: Whole encoding
                        ENCSPU_ENCODE_START: Start divided encoding
                        ENCSPU_ENCODE_CONTINUE: Continue divided encoding
                        ENCSPU_ENCODE_END: End divided encoding
                        Encoding quality. Only effective for EncSPU2().
                        Specify either ENCSPU_ENCODE_MIDDLE_QUALITY or
                        ENCSPU_ENCODE_HIGH_QUALITY.
    char pad4;            System reserved
} ENCSPUENV;
```

Explanation

This structure is used to specify the SPU encode environment attributes for EncSPU() function.

When **ENCSPU_ENCODE_NO_LOOP** is specified for *loop*, *loop_start* is ignored.

See also

[EncSPU\(\)](#)

DecDCTBufSize

Get size of run-level DCT data.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	12/14/98

Syntax

```
long DecDCTBufSize(
u_long *bs)           Pointer to bitstream
```

Explanation

Returns the uncompressed length of the data contained in the Huffman-encoded bitstream pointed to by the *bs* parameter. It does not perform the actual decoding.

When using DecDCTvlc()/DecDCTvlc2() to perform decoding, you must reserve a 1-word header buffer to add to the size obtained by this function.

Return value

Length of uncompressed data in long words (i.e. returns 1000 for a 4000-byte length).

See also

[DecDCTvlc\(\)](#), [DecDCTvlc2\(\)](#)

DecDCTGetEnv

Get current quantization tables and environment data used during MDEC image decoding.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.5	12/14/98

Syntax

`DECDC TENV *DecDCTGetEnv(
DECDC TENV *env)` Pointer to decoding environment

Explanation

Returns the current decoding environment to *env*.

Return value

Address of *env*.

See also

`DecDCTPutEnv()`

DecDCTin

Begin decoding RLE-encoded MDEC image data.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	12/14/98

Syntax

```
void DecDCTin(
  unsigned long *runlevel,  Pointer to input runlevel
  long mode)               Decode mode
```

Explanation

Begins decoding the RLE-encoded MDEC image data at the address specified by *runlevel*. A maximum of 128k may be decoded at a time. The resulting image data is retrieved by the DecDCTout() function.

Bit 0 of the *mode* parameter controls the depth of the output pixels: 0 = 16-bit direct color; 1 = 24-bit direct color. In 16-bit mode, bit 1 of *mode* is the STP bit that determines bit 15 of the pixel.

The image data produced is raw pixel data without any header information. The width and height of the image is not maintained; the application or a higher level structure (such as the STR format) must maintain such information.

Data decoded from a single DecDCTin() call may be read using multiple DecDCTout() calls, or the data created by multiple DecDCTin() calls may be read using a single DecDCTout() call.

DecDCTin() is non-blocking. To detect when execution of the primitive list is complete, use DecDCTinSync() or install a callback routine with DecDCTinCallback(). If DecDCTin() is called before a previous DecDCTin() operation has finished, it is blocked until the previous operation is complete.

See also

[DecDCTout\(\)](#), [DecDCTinSync\(\)](#), [DecDCTinCallback\(\)](#)

DecDCTinCallback

Install a callback routine to be called at termination of MDEC transmission.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

```
long DecDCTinCallback(
void (*func)())           Pointer to callback function
```

Explanation

Installs the user-defined callback routine specified by *func*. This routine is called when the data transmission initiated by a DecDCTin() call has been completed. If *func* is 0, any previous callback routine is disabled.

Although the callback is called during an interrupt, it is not an interrupt handler; it should be written as a normal subroutine that is called by the main interrupt handler. Inside the callback, subsequent termination interrupts are masked; therefore, the callback should return as soon as possible.

Return value

A pointer to a previously set callback function.

See also

[DecDCTin\(\)](#)

DecDCTinSync

Detect DecDCTin() termination.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

**long DecDCTinSync(
long mode)** 0: Blocks until termination; 1: Performs only status notification

Explanation

Detects termination of DecDCTin().

Synchronization with DecDCTinSync() must be performed after reading the appropriate amount of decode data with DecDCTout(). When calling this function without using DecDCTout() after DecDCTin(), a timeout occurs and MDEC is reset.

Return value

Image processing subsystem status: 1 if transmission is in process and 0 if transmission is not being performed.

See also

[DecDCTin\(\)](#), [DecDCTout\(\)](#)

DecDCTout

Receive decoded data from the image processing subsystem.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	12/14/98

Syntax

```
void DecDCTout(
  unsigned long *cell,    Pointer to decoded image data
  long size)             Received data size (long words)
```

Explanation

The RLE-encoded MDEC image data previously specified in a DecDCTin() call is decoded and stored in the buffer specified by the *cell* parameter. The amount of data is specified in long words by *size* (e.g. size=1000 to transfer 4000 bytes of data). Multiple calls to DecDCTout() may be made to retrieve image data.

You must specify a *size* value that is the same as or smaller than the available decoded data. If there is more data available than is read by one DecDCTout() call, additional calls must be made to avoid MDEC transmission deadlocks.

The decoded image is output one 16 x 16 macroblock at a time. *size* must be a multiple of the total macroblock size for the current decoding mode. If decoding to 16-bit, a macroblock is 128 words. If decoding to 24-bit, the macroblock length is 192 words.

DecDCTout() is non-blocking. To detect when execution is complete, use DecDCToutSync() or install a callback routine with DecDCToutCallback(). If a DecDCTout() call is executed before a previous one has finished, the transmission is blocked until the previous operation is complete.

See also

[DecDCTin\(\)](#), [DecDCToutSync\(\)](#), [DecDCToutCallback\(\)](#)

DecDCToutCallback

Install a callback routine to be called at termination of MDEC transmission.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	12/14/98

Syntax

```
long DecDCToutCallback(
long (*func)())          Pointer to callback function address
```

Explanation

Installs the user-defined callback routine specified by *func*. This routine is called when the data transmission initiated by a DecDCTout() call has been completed. If *func* is 0, any previous callback routine is disabled.

Although the specified function is called during an interrupt, it is not an interrupt handler; it should be written as a normal subroutine that is called by the main interrupt handler. Inside the callback, subsequent transmission termination interrupts are masked; therefore, the callback should return as soon as possible.

Return value

A pointer to the previously set callback function.

See also

[DecDCTout\(\)](#)

DecDCToutSync

Detect termination of DecDCTout().

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	12/14/98

Syntax

long DecDCToutSync(
long *mode*) 0: blocks until termination; 1: performs only status notification

Explanation

Detects termination of DecDCTout().

Return value

Image processing subsystem status: 1 if reception is in progress and 0 if reception is not being performed.

See also

[DecDCTout\(\)](#)

DecDCTPutEnv

Set image-processing-subsystem environment.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.5	12/14/98

Syntax

`DECDCTENV *DecDCTPutEnv(
DECDCTENV *env)` Pointer to decoding environment

Explanation

Sets the quantization tables and environment data used during the reverse-quantization step of the MDEC decoding process.

Return value

Address of *env*.

See also

`DecDCTGetEnv()`

DecDCTReset

Initialize image processing subsystem.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	2.x	12/14/98

Syntax

```
void DecDCTReset(  
long mode)
```

0: Initializes all internal states

1: Discontinues only current decoding; does not affect internal states

Explanation

Resets the image processing subsystem.

Processing time is longer for *mode* 0 than for *mode* 1 because internal tables are initialized.

DecDCTvlc

Decode Huffman-compressed MDEC image data.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	<i>2.x</i>	<i>12/14/98</i>

Syntax

int DecDCTvlc(
 u_long *bs, Input bitstream
 u_long *buf) Output runlevel

Explanation

Builds the run-level intermediate format in *buf* by decoding the bitstream *bs*. If runlevel data exceeds the value specified in DecDCTvlcSize(), DecDCTvlc() is interrupted and returns control to the application. The interrupted VLC decode can be restarted by executing DecDCTvlc (0,0). With *buf*, the 1 word area added to the header buffer in DecDCTBufSize() must be reserved in advance.

This is a blocking function.
This function is only the first stage of decoding an MDEC image. The Huffman-encoded bitstream must always be decoded using DecDCTvlc() before DecDCTin() is executed.
A partial result run level cannot be provided as DecDCTin() input.

Return value

- | | |
|----|--|
| 0 | Decoding for all bit stream is successfully completed. |
| 1 | Returned with some bit stream left non-decoded. |
| -1 | Decode failed. |

See also

[DecDCTvlc2\(\)](#), [DecDCTin\(\)](#), [DecDCTBufSize\(\)](#), [DecDCTvlcSize\(\)](#)

DecDCTvlc2

Decode VLC.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.7	12/14/98

Syntax

int DecDCTvlc2(
 u_long *bs, Input bit stream
 u_long *buf, Output run level
 DECDCCTAB table) VLC table

Explanation

Builds the run-level intermediate format in *buf* by decoding the bitstream *bs* using the *table*. When the run level data exceeds the value specified in DecDCTvlcSize2(), DecDCTvlc2() is suspended and control is returned to the application. The suspended VLC decoding process can be restarted by executing DecDCTvlc2(0, 0, *table*). With *buf*, the 1-word area added to the header buffer in DecDCTBufSize() must be reserved in advance.

This is a blocking function. This function is only the first stage of decoding an MDEC image. The Huffman-encoded bitstream must always be decoded using DecDCTvlc() before DecDCTin() is executed. A partial result run level cannot be provided as DecDCTin() input. The VLC table should be decoded in advance using DecDCTBuild().

Return value

- | | |
|----|--|
| 0 | Decoding for all bit stream is successfully completed. |
| 1 | Returned with some bit stream left non-decoded. |
| -1 | Decode failed. |

See also

[DecDCTvlcSize2\(\)](#), [DecDCTin\(\)](#), [DecDCTvlcBuild\(\)](#), [DecDCTBufSize\(\)](#)

DecDCTvlcBuild

Build the VLC table.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.7	12/14/98

Syntax

**void DecDCTvlcBuild(
u_short *table)** VLC Buffer

Explanation

Builds the VLC table that will be used for DecDCTvlc2(). The size of the VLC table to be built can be obtained using sizeof (DECDCTTAB). See libpress.h for the definition of DECDCTTAB.

The VLC table is held in a compressed (4KB) format and only when a movie is playing is it released to the work area and used in its decompressed form (64 KB).

See also

[DecDCTvlc2\(\)](#)

DecDCTvlcSize

Set maximum amount of data returned by a single call to DecDCTvlc().

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.2	12/14/98

Syntax

```
int DecDCTvlcSize(
int size)           Maximum value of a decoded runlevel (long word)
```

Explanation

Sets the maximum number of long words that DecDCTvlc() can return. Subsequent calls to DecDCTvlc() halt after decoding *size* long words. If *size* is zero, DecDCTvlc() decodes the entire bitstream regardless of length.

This allows your program to make multiple calls to DecDCTvlc() to decode a bitstream in chunks using a smaller buffer size.

This is a blocking function. A bitstream must be converted to run-levels by DecDCTvlc() before executing DecDCTin().

Return value

Previously set buffer size.

Example:

```
/* Decoding the first VLC_SIZE word in VLC */
DecDCTvlcSize (VLC_SIZE);
isvlcLeft = DecDCTvlc (next, dec.vlcbuf[dec.vlcid]);
/* Waiting for data to be completed */
do {
    /* Decoding the remaining VLC_SIZE words in VLC */
    if (isvlcLeft) {
        isvlcLeft = DecDCTvlc (0, 0);
        FntPrint ("%d, ", VSync (1));
    }
    /* Application code is here */
} while (isvlcLeft || isEndOfFlame == 0);
isEndOfFlame = 0;
```

See also

[DecDCTvlc\(\)](#), [DecDCTin\(\)](#)

DecDCTvlcSize2

Set maximum size of single VLC decoding process.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.7	12/14/98

Syntax

**int DecDCTvlcSize2(
int size)** Maximum value of a decoded runlevel (long word)

Explanation

Sets the maximum size of bitstream that can be decoded per decoding process. DecDCTvlc2() suspends the decoding process when decoding the first block after the number of words specified by *size*. If *size* is 0 (the default), the decoding process is not suspended.

Since this is a blocking function, the bit stream must be converted to a run level by DecDCTvlc2() before executing DecDCTin().

Return value

Maximum run level set immediate before.

See also

[DecDCTvlc2\(\)](#), [DecDCTin\(\)](#)

EncSPU

Encode 16-bit PCM data into PlayStation waveform format.

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	3.6	12/14/98

Syntax

long EncSPU(

ENCSPUENV **es_env*) SPU encode environment attribute structure

Explanation

Encodes the 16-bit straight PCM data specified by *es_env.src* into PlayStation waveform data (VAG, without header information) and returns the encoded data in *es_env.dest*.

16-bit straight PCM data size in *es_env.size* is specified in byte units. When creating looping waveforms, specify *es_env.loop* as ENCSPU_ENCODE_LOOP and specify *es_env.loop_start* as the *es_env.src* internal PCM data loop start point in byte units.

If *es_env.loop_start* is not a multiple of 56 (28 samples), the loop start point is set to the next lower multiple of 56. If it is not looping waveform, specify *es_env.loop* as ENCSPU_ENCODE_NO_LOOP.

To ensure proper compression of different PCM endian formats, specify *es_env.byte_swap* as ENCSPU_ENCODE_ENDIAN_BIG (16-bit big endian) or ENCSPU_ENCODE_ENDIAN_LITTLE (16-bit little endian).

Whole/Divided encoding specifications are performed by specifying an attribute to *es_env.proceed*:

<i>es_env.proceed</i>	Encoding Specifications
ENCSPU_ENCODE_WHOLE	Whole encoding
ENCSPU_ENCODE_START	Start divided encoding
ENCSPU_ENCODE_CONTINUE	Continue divided encoding
ENCSPU_ENCODE_END	End divided encoding

When *es_env.proceed* is set to something other than ENCSPU_ENCODE_WHOLE, the area is divided, in other words, the following encoding is performed only in the area specified from *es_env.src* to *es_env.size*:

- Encoding by ENCSPU_ENCODE_START of the area including the start block
- Continued encoding by ENCSPU_ENCODE_CONTINUE of the intermediate area
- Encoding by ENCSPU_ENCODE_END of the section including the end block

If *es_env.size* is not a multiple of 56 (28 samples), the data is padded with zeroes until it is. This could cause the generated waveform to be discontinuous; to maintain continuity, perform a divided encode on the data with *es_env.size* equal to a multiple of 56.

If *es_env.proceed* is set to ENCSPU_ENCODE_WHOLE, the waveform is padded with zeroes to make *es_env.size* a multiple of 56, and waveform encoding is performed all at once.

To use the scratchpad as the workspace, specify *es_env.work* as the scratchpad address; use 168 bytes from the specified address. If *es_env.work* is set to NULL, the automatic variables are used internally.

When *loop* is specified as ENCSPU_ENCODE_NO_LOOP, *loop_start* will be ignored.

Return value

The data size of the encoded waveform (VAG).

ENC_ENCODE_ERROR is returned when an encoding error occurs.

Notes

Be aware that processing speed is faster with EncSPu than EncSPU2, but sound quality is poorer.

EncSPU2

Encode 16-bit PCM data into PlayStation waveform format (EncSPU2 high quality sound version)

Library	Header File	Introduced	Documentation Date
<i>libpress.lib</i>	<i>libpress.h</i>	4.6	8/27/99

Syntax

long EncSPU2(
ENCSPUENV *es_env) SPU encode environment attribute structure

Explanation

Encodes the 16-bit straight PCM data specified by *es_env.src* into PlayStation waveform data (VAG, without header information) and returns the encoded data in *es_env.dest*.

16-bit straight PCM data size in *es_env.size* is specified in byte units. When creating looping waveforms, specify *es_env.loop* as ENCSPU_ENCODE_LOOP and specify *es_env.loop_start* as the *es_env.src* internal PCM data loop start point in byte units.

If *es_env.loop_start* is not a multiple of 56 (28 samples), the loop start point is set to the next lower multiple of 56. If it is not looping waveform, specify *es_env.loop* as ENCSPU_ENCODE_NO_LOOP.

To ensure proper compression of different PCM endian formats, specify *es_env.byte_swap* as ENCSPU_ENCODE_ENDIAN_BIG (16-bit big endian) or ENCSPU_ENCODE_ENDIAN_LITTLE (16-bit little endian).

Whole/Divided encoding specifications are performed by specifying an attribute to *es_env.proceed*:

<i>es_env.proceed</i>	Encoding Specifications
ENCSPU_ENCODE_WHOLE	Whole encoding
ENCSPU_ENCODE_START	Start divided encoding
ENCSPU_ENCODE_CONTINUE	Continue divided encoding
ENCSPU_ENCODE_END	End divided encoding

When *es_env.proceed* is set to something other than ENCSPU_ENCODE_WHOLE, the area is divided, in other words, the following encoding is performed only in the area specified from *es_env.src* to *es_env.size*:

- Encoding by ENCSPU_ENCODE_START of the area including the start block
- Continued encoding by ENCSPU_ENCODE_CONTINUE of the intermediate area
- Encoding by ENCSPU_ENCODE_END of the section including the end block

If *es_env.size* is not a multiple of 56 (28 samples), the data is padded with zeroes until it is. This could cause the generated waveform to be discontinuous; to maintain continuity, perform a divided encode on the data with *es_env.size* equal to a multiple of 56.

If *es_env.proceed* is set to ENCSPU_ENCODE_WHOLE, the waveform is padded with zeroes to make *es_env.size* a multiple of 56, and waveform encoding is performed all at once.

To use the scratchpad as the workspace, specify *es_env.work* as the scratchpad address; use 168 bytes from the specified address. If *es_env.work* is set to NULL, the automatic variables are used internally.

However, when *es_env_quality* is set to ENCSPU_ENCODE_HIGH_QUALITY, only NULL can be specified. With regard to quality and speed, when *es_env_quality* is set to ENCSPU_ENCODE_MIDDLE_QUALITY, encoding is performed with an emphasis on speed rather than quality. When *es_env_quality* is set to ENCSPU_ENCODE_HIGH_QUALITY, though, the emphasis is placed on quality rather than speed.

When *loop* is specified as ENCSPU_ENCODE_NO_LOOP, *loop_start* will be ignored.

Return value

The data size of the encoded waveform (VAG).

ENC_ENCODE_ERROR is returned when an encoding error occurs.

Notes

Be aware that sound quality is better with EncSPU2 than EncSPU, but processing is slower.

Chapter 7: Basic Graphics Library

Table of Contents

Structures

DISPENV	7-5
DRAWENV	7-6
DR_AREA	7-7
DR_ENV	7-8
DR_LOAD	7-9
DR_MODE	7-10
DR_MOVE	7-11
DR_OFFSET	7-12
DR_STP	7-13
DR_TPAGE	7-14
DR_TWIN	7-15
LINE_F2, LINE_F3, LINE_F4	7-16
LINE_G2, LINE_G3, LINE_G4	7-17
POLY_F3, POLY_F4	7-19
POLY_FT3, POLY_FT4	7-20
POLY_G3, POLY_G4	7-22
POLY_GT3, POLY_GT4	7-23
RECT	7-25
RECT32	7-26
SPRT	7-27
SPRT_8, SPRT_16	7-28
TILE	7-29
TILE_1, TILE_8, TILE_16	7-30
TIM_IMAGE	7-31
TMD_PRIM	7-32

Functions

AddPrim, addPrim	7-33
AddPrims, addPrims	7-34
BreakDraw	7-35
CatPrim, catPrim	7-36
CheckPrim	7-37
ClearImage	7-38
ClearImage2	7-39
ClearOTag	7-40
ClearOTagR	7-41
ContinueDraw	7-42
DrawOTag	7-43
DrawOTag2	7-44
DrawOTagEnv	7-45
DrawOTagIO	7-46
DrawPrim	7-47
DrawSync	7-48
DrawSyncCallback	7-49
DumpClut, dumpClut	7-50
DumpDispEnv	7-51
DumpDrawEnv	7-52
DumpOTag	7-53

DumpTPage, dumpTPage	7-54
FntFlush	7-55
FntLoad	7-56
FntOpen	7-57
FntPrint	7-58
GetClut, getClut	7-59
GetDispEnv	7-60
GetDrawArea	7-61
GetDrawEnv	7-62
GetDrawEnv2	7-63
GetDrawMode	7-64
GetDrawOffset	7-65
GetGraphDebug	7-66
GetODE	7-67
GetTexWindow	7-68
GetTimSize	7-69
GetTPage, getTPage	7-70
IsEndPrim, isendprim	7-71
IsIdleGPU	7-72
KanjiFntClose	7-73
KanjiFntFlush	7-74
KanjiFntOpen	7-75
KanjiFntPrint	7-76
Krom2Tim	7-77
LoadClut	7-78
LoadClut2	7-79
LoadImage	7-80
LoadImage2	7-81
LoadTPage	7-82
MargePrim	7-83
MovelImage	7-84
MovelImage2	7-85
NextPrim, nextPrim	7-86
OpenTIM	7-87
OpenTMD	7-88
PutDispEnv	7-89
PutDrawEnv	7-90
ReadTIM	7-91
ReadTMD	7-92
ResetGraph	7-93
SetDefDispEnv	7-94
SetDefDrawEnv	7-95
SetDispMask	7-96
SetDrawArea	7-97
SetDrawEnv	7-98
SetDrawLoad	7-99
SetDrawMode, setDrawMode	7-100
SetDrawMove	7-101
SetDrawOffset	7-102
SetDrawStp	7-103
SetDrawTPage, setDrawTPage	7-104
SetDumpFnt	7-105
SetGraphDebug	7-106

SetLineF2, SetLineF3, SetLineF4; setLineF2, setLineF3, setLineF4; SetLineG2, SetLineG3, SetLineG4; setLineG2, setLineG3, setLineG4	7-107
SetPolyF3, SetPolyF4; setPolyF3, setPolyF4; SetPolyG3, SetPolyG4; setPolyG3, setPolyG4; SetPolyGT3, SetPolyGT4; setPolyGT3, setPolyGT4	7-108
SetSemiTrans, setSemiTrans	7-109
SetShadeTex, setShadeTex	7-110
SetSprt, SetSprt8, SetSprt16; setSprt, setSprt8, setSprt16	7-111
SetTexWindow	7-112
SetTile, SetTile1, SetTile8, SetTile16; setTile, setTile1, setTile8, setTile16	7-113
StoreImage	7-114
StoreImage2	7-115
TermPrim, termPrim	7-116
VSync	7-117
VSyncCallback	7-118

Macros

addVector	7-119
applyVector	7-120
copyVector	7-121
dumpMatrix	7-122
dumpRECT	7-123
dumpVector	7-124
dump...	7-125
setClut	7-126
setRECT	7-127
setRGB0, setRGB1, setRGB2, setRGB3	7-128
setTPage	7-129
setUV0, setUV3, setUV4	7-130
setUWH	7-131
setVector	7-132
setWH	7-133
setXY0, setXY2, setXY3, setXY4	7-134
setXYWH	7-135

Structures

DISPENV

Display environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct DISPENV {
    RECT disp;           Display area within frame buffer. Width: 256, 320, 384, 512, or 640. Height:
                        240 or 480.
    RECT screen;         Output screen display area. It is calculated without regard to the value of
                        disp, using the standard monitor screen upper left-hand point (0, 0) and
                        lower right-hand point (256, 240).
    u_char isinter;      Interlace mode flag. 0: non-interlace; 1: interlace
    u_char isrgb24;      24-bit mode flag. 0: 16-bit mode; 1: 24-bit mode
    u_char pad0, pad1;   Reserved by system
};
```

Explanation

Specifies display parameters for screen display mode, frame buffer display value, and so on.

See also

[DumpDispEnv\(\)](#), [GetDispEnv\(\)](#), [PutDispEnv\(\)](#), [SetDefDispEnv\(\)](#)

DRAWENV

Drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

struct DRAWENV {

RECT *clip*; Drawing area. Drawing is restricted to the area specified by *clip*. It must be within the area (0, 0) - (1023, 511).

short *ofs[2]*; The offsets *ofs[0]* and *ofs[1]* are added to the X and Y values, respectively, of all primitives before drawing. **Note:** Addresses after adding offsets are wrapped around at (-1024, -1024) - (1023, 1023).

RECT *tw*; Texture window. Specifies a rectangle inside the texture page, to be used for drawing textures.

u_short *tpage*; Initial value of texture page

u_char *dtd*; Dithering processing flag. 0: off; 1: on

u_char *dfe*; 0: drawing to display area is blocked
1: drawing to display area is permitted

u_char *isbg*; 0: Does not clear drawing area when drawing environment is set.
1: Paints entire clip area with brightness values (*r0*, *g0*, *b0*) when drawing environment is set.

u_char *r0*, *g0*, *b0*; Background color. Valid only when *isbg* is 1.

DR_ENV *dr_env*; System reserved

};

Explanation

Sets basic drawing parameters, such as drawing offset and drawing clip area.

The GPU uses 8 bits for R, G, B internally; when writing to the frame buffer, each value is reduced to 5 bits. When *dtd* is ON, a 4x4 dither matrix is used as follows:

```
i = 8 bit brightness value + 1/2 * D - 4
D = Dither matrix [X%4][Y%4]
```

Table 7-1: 4x4 Dither Matrix

0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

5 bit brightness value = 1 >> 3

The values which may be specified for the texture window are restricted to the following combinations:

Table 7-2

tw.w, tw.x						
tw.w	0 (=256)	8	16	32	64	128
tw.x	0	Multiple of 8	Multiple of 16	Multiple of 32	Multiple of 64	Multiple of 128
tw.h, tw.y						
tw.h	0 (=256)	8	16	32	64	128
tw.y	0	Multiple of 8	Multiple of 16	Multiple of 32	Multiple of 64	Multiple of 128

See also

[DrawOTagEnv\(\)](#), [DumpDrawEnv\(\)](#), [GetDrawEnv\(\)](#), [PutDrawEnv\(\)](#), [SetDefDrawEnv\(\)](#), [SetDrawEnv\(\)](#)

DR_AREA

Drawing area change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Structure

```

struct DR_AREA {
    u_long *tag;           Pointer to the next primitive in primitive list
    u_long code[2];        New drawing area information specified by SetDrawArea()
};

```

Explanation

Modifies the drawing area of the current drawing environment while a primitive list is being drawn. Use `SetDrawArea()` to set the contents of this primitive.

See also

[GetDrawArea\(\)](#), [SetDrawArea\(\)](#)

DR_ENV

Drawing environment modification primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct DR_ENV {
    u_long *tag;           Pointer to the next primitive in primitive list
    u_long code[15];       New drawing environment information specified by SetDrawEnv()
};
```

Explanation

Changes the drawing environment ([DRAWENV](#)) while a primitive list is being drawn. Use `SetDrawEnv()` to specify the new DRAWENV parameters.

This primitive affects only the drawing environment, not the display environment (see [DISPENV](#)). The entire drawing environment may be changed using this primitive; see also the [DR_MODE](#) primitive, which sets a subset of the drawing environment.

See also

[SetDrawEnv\(\)](#), [PutDrawEnv\(\)](#)

DR_LOAD

Load Image primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.4	12/14/98

Structure

```
typedef struct {
    u_long *tag;           Pointer to next primitive (reserved)
    u_long code[3];        Primitive ID
    u_long p[13];          Transfer data
} DR_LOAD;
```

Explanation

Transfers data below array *p* to the frame buffer. As with LoadImage(), semitransparent/ transparent color control is not performed. Also, there is no dependence on the drawing environment.

Maximum data transfer amount is 12 words (24 pixels).

See also

[LoadImage\(\)](#), [SetDrawLoad\(\)](#)

DR_MODE

Drawing mode modification primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    u_long *tag;           Pointer to the next primitive in primitive list
    u_long code[2];        New drawing environment information as specified by
                           SetDrawMode()
} DR_MODE;
```

Explanation

Changes the texture page, texture window, dithering flag, and drawing flag parameters of the current drawing environment while a primitive list is being drawn. See the *tpage*, *tw*, *dtd*, and *dfe* members of the [DRAWENV](#) structure for more information. Use [SetDrawMode\(\)](#) to specify the parameters to be used.

See also

[SetDrawMode\(\)](#), [GetDrawMode\(\)](#)

DR_MOVE

Rectangle copy primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.2	12/14/98

Structure

```
typedef struct {
    u_long tag;           Hook to the next primitive (reserved)
    u_long code[5];       Primitive ID
} DR_MOVE;
```

Explanation

Copies a rectangle. Speed is the same as MoveImage().

Unlike the 16-bit [SPRT](#) primitive, semitransparent/transparent color control is not carried out. Also, transfer does not depend on the drawing environment.

See also

[MoveImage\(\)](#), [MoveImage2\(\)](#), [SetDrawMove\(\)](#)

DR_OFFSET

Drawing offset modification primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    u_long *tag;           Pointer to the next primitive in primitive list
    u_long code[2];        New drawing offset information specified by SetDrawOffset()
} DR_OFFSET;
```

Explanation

Changes the drawing offset parameters of the current drawing environment while a primitive list is being drawn. See the *ofs* member of the [DRAWENV](#) structure for more information. Use [SetDrawOffset\(\)](#) to specify the parameters to be used.

See also

[GetDrawOffset\(\)](#), [SetDrawOffset\(\)](#)

DR_STP

STP bit updated primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	12/14/98

Structure

```
typedef struct DR_STP {
    u_long tag;           Pointer to the next primitive in primitive list (reserved)
    u_long code[2];       Primitive ID
} DR_STP;
```

Explanation

Updates the STP bit during drawing. Use `SetDrawStp()` to set the contents of this primitive.

See also

[SetDrawStp\(\)](#)

DR_TPAGE

Texture page change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.5	12/14/98

Structure

```
typedef struct {
    u_long *tag;           Pointer to the next primitive in primitive list
    u_long code[2];        New texture page information specified by SetDrawTPage()
} DR_TPAGE;
```

Explanation

Changes the texture page parameter of the current drawing environment while a primitive list is being drawn. See the *tpage* member of the [DRAWENV](#) structure for more information. Use SetDrawTPage() to specify the parameters to be used.

See also

[SetDrawTPage\(\)](#)

DR_TWIN

Texture window change primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    u_long *tag;           Pointer to the next primitive in primitive list
    u_long code[2];        New texture window information specified by
                           SetDrawTexWindow()
} DR_TWIN;
```

Explanation

Changes the texture window of the current drawing environment while a primitive list is being drawn. See the *tw* member of the [DRAWENV](#) structure for more information. Use `SetTexWindow()` to specify the parameters to be used.

See also

[GetTexWindow\(\)](#), [SetTexWindow\(\)](#)

LINE_F2, LINE_F3, LINE_F4

One flat-shaded non-connecting line/ Two flat-shaded connected lines/ Three flat-shaded connected lines.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```

struct LINE_F2 {
    u_long *tag;           Pointer to the next primitive (reserved)
    u_char r0, g0, b0;    RGB color specified by straight line
    u_char code;          Primitive ID
    short x0, y0, x1, y1;  Coordinate of vertices forming straight lines
};

```

```

struct LINE_F3 {
    u_long *tag;           Pointer to the next primitive (reserved)
    u_char r0, g0, b0;    RGB color specified by straight line
    u_char code;          Primitive ID
    short x0, y0, x1, y1, x2, y2;  Coordinate of vertices forming straight lines
    u_long pad;           Reserved
};

```

```

struct LINE_F4 {
    u_long *tag;           Pointer to the next primitive (reserved)
    u_char r0, g0, b0;    RGB color specified by straight line
    u_char code;          Primitive ID
    short x0, y0, x1, y1, x2, y2, x3, y3;  Coordinate of vertices forming straight lines
    u_long pad;           Reserved
};

```

Explanation

LINE_F2 draws a non-connecting line linking (x0, y0) - (x1, y1) with the RGB color specified by (r0, g0, b0).

LINE_F3 draws 2 connecting lines linking (x0, y0) - (x1, y1) - (x2, y2) with the RGB color specified by (r0, g0, b0).

LINE_F4 draws 3 connecting lines linking (x0, y0) - (x1, y1) - (x2, y2) - (x3, y3), with the RGB color specified by (r0, g0, b0).

See also

[SetLineF2\(\)](#)

LINE_G2, LINE_G3, LINE_G4

One Gouraud-shaded non-connecting line/ Two Gouraud-shaded connected lines/ Three Gouraud-shaded connected lines

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```

struct LINE_G2 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    u_char r1, g1, b1;     RGB color values
    u_char p1;             Primitive ID (reserved)
    short x1, y1;          Vertex coordinates
};

```

```

struct LINE_G3 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    u_char r1, g1, b1;     RGB color values
    u_char p1;             Primitive ID (reserved)
    short x1, y1;          Vertex coordinates
    u_char r2, g2, b2;     RGB color values
    u_char p2;             Primitive ID (reserved)
    short x2, y2;          Vertex coordinates
    u_long pad;            Reserved
};

```

```

struct LINE_G4 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    u_char r1, g1, b1;     RGB color values
    u_char p1;             Primitive ID (reserved)
    short x1, y1;          Vertex coordinates
    u_char r2, g2, b2;     RGB color values
    u_char p2;             Primitive ID (reserved)
    short x2, y2;          Vertex coordinates
    u_char r3, g3, b3;     RGB color values
    u_char p3;             Primitive ID (reserved)
    short x3, y3;          Vertex coordinates
    u_long pad;            Reserved
};

```

Explanation

LINE_G2 draws a non-connecting line linking $(x0, y0) - (x1, y1)$ in such a way that its vertices have the RGB color specified by $(r0, g0, b0) - (r1, g1, b1)$, and perform Gouraud shading at the same time.

LINE_G3 draws connecting lines linking $(x0, y0) - (x1, y1) - (x2, y2)$ in such a way that their vertices have the RGB color specified by $(r0, g0, b0) - (r1, g1, b1) - (r2, g2, b2)$, and perform Gouraud shading at the same time.

7-18 Basic Graphics Library Structures

LINE_G4 draws connecting lines linking $(x0, y0) - (x1, y1) - (x2, y2) - (x3, y3)$ in such a way that their vertices have the RGB color specified by $(r0, g0, b0) - (r1, g1, b1) - (r2, g2, b2) - (r3, g3, b3)$ and perform Gouraud shading at the same time.

See also

[SetLineF2\(\)](#)

POLY_F3, POLY_F4

Flat-shaded, non-textured mapped triangle/ Flat-shaded, non-textured mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct POLY_F3 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    short x1, y1;          Vertex coordinates
    short x2, y2;          Vertex coordinates
};
```

```
struct POLY_F4 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    short x1, y1;          Vertex coordinates
    short x2, y2;          Vertex coordinates
    short x3, y3;          Vertex coordinates
};
```

Explanation

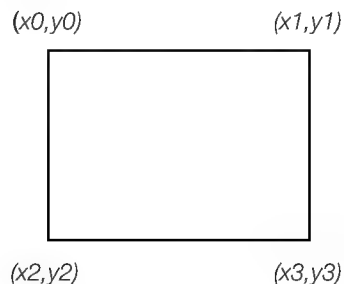
POLY_F3 paints the area demarcated by $(x0, y0) - (x1, y1) - (x2, y2)$ using RGB color specified by $(r0, g0, b0)$.

POLY_F4 paints the area demarcated by $(x0, y0) - (x1, y1) - (x3, y3) - (x2, y2)$ using RGB color specified by $(r0, g0, b0)$.

The address where a picture is actually drawn is equivalent to the value of $x0$ - $x3$ to which the offset value specified by the drawing environment is added. What is drawn is clipped according to the clip area (quadrilateral area) specified by the drawing environment.

If the polygon has a width greater than 1024 and a height greater than 512, all of it is clipped. In the case of a quadrilateral primitive, the corners are specified in the order shown below.

Figure 7-1



See also

[SetPolyF3\(\)](#)

POLY_FT3, POLY_FT4

Flat-shaded, texture-mapped triangle/ Flat-shaded, texture-mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure**struct POLY_FT3 {**

u_long <i>tag</i> ;	Pointer to the next primitive
u_char <i>r0, g0, b0</i> ;	RGB color values
u_char <i>code</i> ;	Primitive ID (reserved)
short <i>x0, y0</i> ;	Vertex coordinates
u_char <i>u0, v0</i> ;	Texture coordinates
u_short <i>clut</i> ;	CLUT ID (color-look-up table for 4-bit/8-bit mode only)
short <i>x1, y1</i> ;	Vertex coordinates
u_char <i>u1, v1</i> ;	Texture coordinates
u_short <i>tpage</i> ;	Texture page ID
short <i>x2, y2</i> ;	Vertex coordinates
u_char <i>u2, v2</i> ;	Texture coordinates
u_short <i>pad1</i> ;	Reserved by the system

};

struct POLY_FT4 {

u_long <i>tag</i> ;	Pointer to the next primitive
u_char <i>r0, g0, b0</i> ;	RGB color values
u_char <i>code</i> ;	Primitive ID (reserved)
short <i>x0, y0</i> ;	Vertex coordinates
u_char <i>u0, v0</i> ;	Texture coordinates
u_short <i>clut</i> ;	CLUT ID (color-look-up table for 4-bit/8-bit mode only)
short <i>x1, y1</i> ;	Vertex coordinates
u_char <i>u1, v1</i> ;	Texture coordinates
u_short <i>tpage</i> ;	Texture page ID
short <i>x2, y2</i> ;	Vertex coordinates
u_char <i>u2, v2</i> ;	Texture coordinates
u_short <i>pad1</i> ;	Reserved by the system
short <i>x3, y3</i> ;	Vertex coordinates
u_char <i>u3, v3</i> ;	Texture coordinates
u_short <i>pad2</i> ;	Reserved by the system

};

Explanation

POLY_FT3 draws an area demarcated by (x0, y0) - (x1, y1) - (x2, y2) while mapping the area demarcated by (u0, v0) - (u1, v1) - (u2, v2) in the texture pattern on the texture page *tpage*.

POLY_FT4 draws an area demarcated by (x0, y0) - (x1, y1) - (x3, y3) - (x2, y2) while mapping the area demarcated by (u0, v0) - (u1, v1) - (u3, v3) - (u2, v2) in the texture pattern on the texture page *tpage*.

The actual brightness value for drawn graphics are obtained by multiplying the RGB color values from the texture pattern by the RGB color values given by *r0, g0, b0*.

The texture coordinates are the coordinates (0 to 255) inside the texture page corresponding to the vertices of the triangle to be drawn. if the texture mode is 4-bit or 8-bit, the texture coordinates and the actual frame buffer address are not 1-to-1.

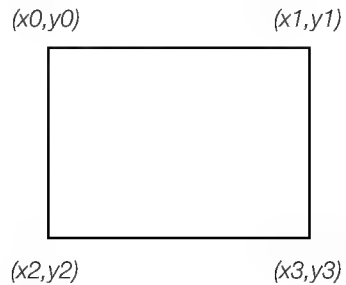
Texture page ID is given to *tpage*. Using GetTPage(), the texture page ID is obtained from the address (x, y) of the buffer frame where the texture page is located.

A texture using CLUT gives CLUT ID to be set in *clut*. Using GetClut(), CLUT ID is obtained from the address (x, y) of the frame buffer where CLUT is located.

The size of the texture page which can be used by one drawing command is 256 x 256. One primitive can only use one texture page.

In the case of a quadrilateral primitive, the corners are specified in the order shown below. The same applies to designation of (u, v) for a texture map rectangle, and (r, g, b) for a Gouraud shaded rectangle.

Figure 7-2



See also

[GetTPage\(\)](#), [GetClut\(\)](#), [SetPolyF3\(\)](#)

POLY_G3, POLY_G4

Gouraud-shaded, non-textured mapped triangle/ Gouraud-shaded, non-textured mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```

struct POLY_G3 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    u_char r1, g1, b1;     RGB color values
    u_char pad1;           Reserved by the system
    short x1, y1;          Vertex coordinates
    u_char r2, g2, b2;     RGB color values
    u_char pad2;           Reserved by the system
    short x2, y2;          Vertex coordinates
};

```

```

struct POLY_G4 {
    u_long *tag;           Pointer to the next primitive
    u_char r0, g0, b0;     RGB color values
    u_char code;           Primitive ID (reserved)
    short x0, y0;          Vertex coordinates
    u_char r1, g1, b1;     RGB color values
    u_char pad1;           Reserved by the system
    short x1, y1;          Vertex coordinates
    u_char r2, g2, b2;     RGB color values
    u_char pad2;           Reserved by the system
    short x2, y2;          Vertex coordinates
    u_char r3, g3, b3;     RGB color values
    u_char pad3;           Reserved by the system
    short x3, y3;          Vertex coordinates
};

```

Explanation

When drawing while performing Gouraud shading, POLY_G3 paints the area demarcated by $(x0, y0) - (x1, y1) - (x2, y2)$ so that vertex RGB color value may be set to $(r0, g0, b0) - (r1, g1, b1) - (r2, g2, b2)$.

When drawing while performing Gouraud shading, POLY_G4 paints the area demarcated by $(x0, y0) - (x1, y1) - (x3, y3) - (x2, y2)$ so that vertex RGB color value may be set to $(r0, g0, b0) - (r1, g1, b1) - (r3, g3, b3) - (r2, g2, b2)$.

The brightness of triangle-internal pixels is calculated by performing linear interpolation of the RGB color values of the three vertices. (Gouraud shading).

See also

[SetPolyF3\(\)](#)

POLY_GT3, POLY_GT4

Gouraud-shaded, texture-mapped triangle/ Gouraud-shaded, texture-mapped quad.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure**struct POLY_GT3 {**

u_long *tag;	Pointer to the next primitive
u_char r0, g0, b0;	RGB color values
u_char code;	Primitive ID (reserved)
short x0, y0;	Vertex coordinates
u_char u0, v0;	Texture coordinates
u_short clut;	CLUT ID (color-look-up table for 4-bit/8-bit mode only)
u_char r1, g1, b1;	RGB color values
u_char p1;	Reserved
short x1, y1;	Vertex coordinates
u_char u1, v1;	Texture coordinates
u_short tpage;	Texture page ID
u_char r2, g2, b2;	RGB color values
u_char p2;	Reserved
short x2, y2;	Vertex coordinates
u_char u2, v2;	Texture coordinates
u_short pad2;	Reserved by the system

};

struct POLY_GT4 {

u_long *tag;	Pointer to the next primitive
u_char r0, g0, b0;	RGB color values
u_char code;	Primitive ID (reserved)
short x0, y0;	Vertex coordinates
u_char u0, v0;	Texture coordinates
u_short clut;	CLUT ID (color-look-up table for 4-bit/8-bit mode only)
u_char r1, g1, b1;	RGB color values
u_char p1;	Primitive ID (reserved)
short x1, y1;	Vertex coordinates
u_char u1, v1;	Texture coordinates
u_short tpage;	Texture page ID
u_char r2, g2, b2;	RGB color values
u_char p2;	Primitive ID (reserved)
short x2, y2;	Vertex coordinates
u_char u2, v2;	Texture coordinates
u_short pad2;	Reserved by the system
u_char r3, g3, b3;	RGB color values
u_char p3;	Primitive ID (reserved)
short x3, y3;	Vertex coordinates
u_char u3, v3;	Texture coordinates
u_short pad3;	Reserved by the system

};

Explanation

POLY_GT3 draws a triangle performing texture mapping and Gouraud shading simultaneously.

POLY_GT4 draws a quadrilateral performing texture mapping and Gouraud shading simultaneously.

The actual RGB color values for the picture are equal to the RGB color values obtained from the texture pattern multiplied by the RGB color values calculated by Gouraud shading.

See also

[SetPolyF3\(\)](#)

RECT

Frame buffer rectangular area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct RECT {
    short x, y;           Top left coordinates of the rectangular area
    short w, h;           Width and height of the rectangular area
};
```

Explanation

Used by several library functions to specify a rectangular area of the frame buffer. Neither negative values, nor values exceeding the size of the frame buffer (1024x512), may be specified.

See also

[ClearImage\(\)](#), [LoadImage\(\)](#), [MoveImage\(\)](#), [StoreImage\(\)](#), [dumpRECT\(\)](#), [setRECT\(\)](#)

RECT32

Rectangular area (32 bit)

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Structure**typedef struct {****int** *x, y*;

Top left coordinates of the rectangular area

int *w, h*;

Width and height of the rectangular area

} RECT32;**Explanation**

Used by several library functions to specify a rectangular area of the frame buffer. Neither negative values, nor values exceeding the size of the frame buffer (1024x512) may be specified.

SPRT

Sprite of any desired size.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct SPRT {
    u_long *tag;           Pointer to next primitive (reserved)
    u_char r0, g0, b0;     RGB color values for sprite
    u_char code;           Primitive code (reserved)
    short x0, y0;          Position of sprite (top left coordinate)
    u_char u0, v0;         Position of sprite texture within the texture page (top left coordinate). u0
                           should be an even number.
    u_short clut;          CLUT ID used (for 4-bit/8-bit mode only)
    short w, h;            Width and height of sprite. w is an even number
};
```

Explanation

Draws a texture-mapped rectangular area. Drawing speed for a SPRT primitive is faster than for a [POLY_FT4](#).

Only even numbers can be specified for *u0* and *w*.

Because the SPRT primitive has no *tpage* parameter, the texture page of the current drawing environment is used. You can change the texture page by inserting a [DR_TPAGE](#) or [DR_MODE](#) primitive into the primitive list before your SPRT primitive.

See also

[SetSprt\(\)](#)

SPRT_8, SPRT_16

8 x 8 fixed size, texture-mapped sprite / 16 x 16 fixed size, texture-mapped sprite.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure**struct SPRT_16 {**

u_long <i>*tag</i> ;	Pointer to next primitive (reserved)
u_char <i>r0, g0, b0</i> ;	RGB color values for sprite
u_char <i>code</i> ;	Primitive code (reserved)
short <i>x0, y0</i> ;	Position of sprite (top left coordinate)
u_char <i>u0, v0</i> ;	Position of sprite texture within the texture page (top left coordinate). <i>u0</i> should be an even number.
u_short <i>clut</i> ;	CLUT ID used (for 4-bit/8-bit mode only)

};

struct SPRT_8 {

u_long <i>*tag</i> ;	Pointer to next primitive (reserved)
u_char <i>r0, g0, b0</i> ;	RGB color values for sprite
u_char <i>code</i> ;	Primitive code (reserved)
short <i>x0, y0</i> ;	Position of sprite (top left coordinate)
u_char <i>u0, v0</i> ;	Position of sprite texture within the texture page (top left coordinate). <i>u0</i> should be an even number.
u_short <i>clut</i> ;	CLUT ID used (for 4-bit/8-bit mode only)

};

Explanation

Draws a sprite with a fixed size of 8 x 8 or 16 x 16. The same result can be obtained if 8 and 16 are designated as the *w* and *h* members of the [SPRT](#) structure.

See also

[SetSprt\(\)](#)

TILE

Tile of any desired size.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct TILE {
    u_long *tag;           Pointer to next primitive (reserved)
    u_char r0, g0, b0;     RGB color values for sprite
    u_char code;           Primitive code (reserved)
    short x0, y0;          Position of sprite (top left coordinate)
    short w, h;            Width and height of sprite. w is an even number
};
```

Explanation

Draws a rectangular area with the specified RGB color value (*r0*, *g0*, *b0*). No texture mapping or shading is done. It is faster than the [POLY_F4](#) primitive.

See also

[SetTile\(\)](#)

TILE_1, TILE_8, TILE_16

1 x 1 fixed-size tile sprite / 8 x 8 fixed-size tile sprite / 16 x 16 fixed-size tile sprite.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
struct TILE_16 {
    u_long *tag;           Pointer to next primitive (reserved)
    u_char r0, g0, b0;     RGB color values for sprite
    u_char code;           Primitive code (reserved)
    short x0, y0;          Position of sprite (top left coordinate)
};
```

```
struct TILE_8 {
    u_long *tag;           Pointer to next primitive (reserved)
    u_char r0, g0, b0;     RGB color values for sprite
    u_char code;           Primitive code (reserved)
    short x0, y0;          Position of sprite (top left coordinate)
};
```

```
struct TILE_1 {
    u_long *tag;           Pointer to next primitive (reserved)
    u_char r0, g0, b0;     RGB color values for sprite
    u_char code;           Primitive code (reserved)
    short x0, y0;          Position of sprite (top left coordinate)
};
```

Explanation

Fixed-size versions of the [TILE](#) primitive. The rectangular area is drawn with the specified RGB color value (*r0*, *g0*, *b0*). No texture mapping or shading is done. These are faster than the [POLY_F4](#) primitive.

See also

[SetTile\(\)](#)

TIM_IMAGE

TIM format image data header.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    u_long mode;           Pixel mode
    RECT *crect;           Pointer to destination rectangle in VRAM for CLUT data
    u_long *caddr;         Pointer to address of CLUT data in main memory
    RECT *prect;           Pointer to destination rectangle in VRAM for texture image data
    u_long *paddr;         Pointer to address of texture image data in main memory
} TIM_IMAGE;
```

Explanation

TIM data header information is acquired by ReadTIM().

crect and *caddr* are assigned a value of zero for TIM having no CLUT.

See also

[ReadTIM\(\)](#)

TMD_PRIM

TMD format model data header.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    u_long id;           TMD primitive ID
    u_char r0, g0, b0, p0; RGB color values of vertex 1 (+ 1-byte pad)
    u_char r1, g1, b1, p1; RGB color values of vertex 2 (+ 1-byte pad)
    u_char r2, g2, b2, p2; RGB color values of vertex 3 (+ 1-byte pad)
    u_char r3, g3, b3, p3; RGB color values of vertex 4 (+ 1-byte pad)
    u_short tpage;       Texture page ID
    u_short clut;         CLUT ID
    u_char u0, v0, u1, v1; Texture vertex coordinates
    u_char u2, v2, u3, v3; Texture vertex coordinates
    SVECTOR x0, x1, x2, x3; Three-dimensional coordinates
    SVECTOR n0, n1, n2, n3; Normal coordinates
    SVECTOR *v_ofs;       Pointer to start coordinates of a vertex array
    SVECTOR *n_ofs;       Pointer to start coordinates of a normal array
    u_short vert0, vert1; Offset to vertex array
    u_short vert2, vert3; Offset to vertex array
    u_short norm0, norm1; Offset to normal array
    u_short norm2, norm3; Offset to normal array
} TMD_PRIM;
```

Explanation

Information on primitives constituting a TMD object. The information is acquired using ReadTMD(). *x0, x1, x3, n0, n1, n3* are used for an independent vertex model. *v_ofs, n_ofs* and *vert0,..vert3, norm0...norm3* are used for a common vertex model.

Some members have no meaning depending on the TMD primitive type.

See also

[ReadTMD\(\)](#)

Functions

AddPrim, addPrim

Register a primitive to the OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void AddPrim (

void **ot*

OT entry

void **p*)

Start address of primitive to be registered

AddPrim(*ot*, *p*)

Macro version of AddPrim()

Explanation

Registers a primitive beginning with the address **p* to the OT entry **ot* in OT table. *ot* is an ordering table or pointer to another primitive.

A primitive may be added to a primitive list only once in the same frame. Attempting to add it multiple times in the same frame results in a corrupted list.

See also

[AddPrims\(\)](#), [CatPrim\(\)](#)

AddPrims, addPrims

Collectively register primitives to the OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void AddPrims(

void **ot*, OT entry

void **p0*, Start address of primitive list

void **p1*) End address of primitive list

AddPrims(*ot*, *p0*, *p1*) Macro version of AddPrims

Explanation

Registers primitives beginning with *p0* and ending with *p1* to the **ot* entry in the OT.

The primitive list is a list of primitives connected by AddPrim() or created by the local ordering table.

See also

[AddPrim\(\)](#)

BreakDraw

Interrupt drawing.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.4	12/14/98

Syntax

u_long *BreakDraw(void)

Explanation

Interrupts drawing after the current polygon is drawn. The return value is the next drawing entry; to resume drawing, pass this value to DrawOTag().

Return value

Next polygon drawing entry.

However, during a DMA transfer outside the OT (such as LoadImage(), etc.) 0xffffffff is returned.

See also

[ContinueDraw\(\)](#), [DrawOTag\(\)](#), [IsIdleGPU\(\)](#)

CatPrim, catPrim

Concatenate primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

**void CatPrim(
void *p0, void *p1)** Starting addresses of primitives to be concatenated

catPrim(p0, p1) Macro version of CatPrim()

Explanation

Links the primitive *p1* to the primitive *p0*.

AddPrim() adds a primitive to a primitive list. CatPrim() simply concatenates two primitives.

Return value

Start address of *p0*.

See also

[AddPrim\(\)](#)

CheckPrim

Check validity of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
long CheckPrim(
char *s,           Pointer to optional character string
u_long *p)        Pointer to primitive
```

Explanation

Checks the validity of the primitive. If the primitive is found to be invalid, a message is printed with the contents of s followed by the type code and length of the primitive. The primitive is not modified in any case.

Return value

0 for a valid primitive; -1 for an invalid primitive.

ClearImage

Clear Frame Buffer at high speed.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	9/1/99

Syntax

```
int ClearImage(
  RECT *rect,           Pointer to rectangular area to be cleared
  u_char r, u_char g, u_char b) Pixel values to be used for clearing
```

Explanation

Sets the rectangular area *rect* in the Frame Buffer to RGB color values (*r*, *g*, *b*).

Because this is a non-blocking function, the end of the operation must be detected using DrawSync() or by installing a callback with DrawSyncCallback(). The drawing area is not affected by the drawing environment (clip/offset).

When the width and height of the rectangular area exceeds (w,h)=(1024,512), only the (w,h)=(1023,511) area is cleared.

When in interlace mode, use ClearImage2() instead.

Return value

Position of this command in the libgpu command queue.

See also

[ClearImage2\(\)](#), [DrawSync\(\)](#), [DrawSyncCallback\(\)](#)

ClearImage2

Clear Frame Buffer at high speed (interlace mode).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	9/1/99

Syntax

int ClearImage2(

RECT *rect, Pointer to rectangular area to be cleared
u_char r, u_char g, u_char b) Pixel values to be used for clearing

Explanation

Sets the rectangular area *rect* in the Frame Buffer to RGB color values (*r*, *g*, *b*).

Although ClearImage() only clears one field when in interlace mode, ClearImage2() clears both fields. However, the DRAWENV.dfe flag will remain high upon completion of this command. This allows drawing to both visible and non-visible scan lines. Therefore, after using this function in interlace mode, DRAWENV.dfe should be set to 0 to avoid drawing to both fields.

Because this is a non-blocking function, the end of the operation must be detected using DrawSync() or by installing a callback routine with DrawSyncCallback(). The drawing area is not affected by the drawing environment (clip/offset).

When the width and height of the rectangular area exceeds (w,h)=(1024,512), only the (w,h)=(1023,511) area is cleared.

Return value

Position of this command in the libgpu command queue.

See also

[ClearImage\(\)](#), [DrawSync\(\)](#), [DrawSyncCallback\(\)](#)

ClearOTag

Initialize an array to a linked list for use as an ordering table.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
u_long *ClearOTag(
u_long *ot,           OT starting pointer
int n)               Number of entries in OT
```

Explanation

Walks the array specified by *ot* and sets each element to be a pointer to the following element, except the last, which is set to a pointer to a special terminator value which the PlayStation® uses to recognize the end of a primitive list. *n* specifies the number entries in the array.

To execute the OT initialized by `ClearOTag()`, call `DrawOTag(ot)`.

See also

[DrawOTag\(\)](#), [ClearOTagR\(\)](#)

ClearOTagR

Initialize an array to a linked list for use as an ordering table.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void ClearOTagR(
u_long *ot,           Head pointer of OT
long n)              Number of entries in OT
```

Explanation

Walks the array specified by *ot* and sets each element to be a pointer to the previous element, except the first, which is set to a pointer to a special terminator value which the PlayStation uses to recognize the end of a primitive list. *n* specifies how many entries are present in the array.

To execute the OT initialized by `ClearOTagR()`, execute `DrawOTag(ot+n-1)`.

See also

[DrawOTag\(\)](#), [ClearOTag\(\)](#)

ContinueDraw

Continue to draw the OT interrupted by BreakDraw()

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void ContinueDraw(

u_long *inst_ot,

Address of interrupting OT

u_long *cont_ot)

Address of drawn OT immediately after drawing inst_ot

Explanation

Immediately executes the OT supplied by *inst_ot* without entering it in the libgpu queue. When the drawing of *inst_ot* is completed, it then draws *cont_ot*. Since the GPU must be in an immediately executable state, ContinueDraw() must be used in combination with routines such as BreakDraw().

This function is used when you wish to draw a specific OT with certain timing and high priority. In such cases, this can be achieved by using BreakDraw() to interrupt the OT being drawn and by executing the return value as *cont_ot*.

See also

[BreakDraw\(\)](#)

DrawOTag

Execute a list of GPU primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
void DrawOTag(
u_long *ot)           Pointer to a linked list of GPU primitives
```

Explanation

Executes the GPU primitives in the linked list *ot*.

`DrawOTag()` is non-blocking. To detect when execution of the primitive list is complete, use `DrawSync()` or install a callback routine with `DrawSyncCallback()`.

See also

[DrawSync\(\)](#), [DrawSyncCallback\(\)](#)

DrawOTag2

Execute a list of GPU primitives (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	12/14/98

Syntax

```
int DrawOTag2(
u_long *p)           Pointer to a linked list of GPU primitives
```

Explanation

Executes the GPU primitives in the linked list *p*. This operation takes place immediately, regardless of what is in the GPU queue, and on completion, processing of the queue is resumed.

When drawing has been suspended using `BreakDraw()` and you want to execute a linked list of GPU primitives using `DrawOTag()`, immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use `DrawOTag2()`.

When drawing is suspended with `BreakDraw()` after `DrawOTag2()` is called, before restarting the drawing with `ContinueDraw()`, it is necessary to confirm the completion of data transfer using `IsIdleGPU()`. This is because `DrawOTag2()` is a non-blocking function.

Return value

0: Normal completion; -1: Abnormal completion.

See also

[BreakDraw\(\)](#), [ContinueDraw\(\)](#), [IsIdleGPU\(\)](#), [DrawOTag\(\)](#)

DrawOTagEnv

Set the drawing environment and draw the primitives registered in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
void DrawOTagEnv(
  u_long *p,           OT start pointer
  DRAWENV *env)        Drawing environment
```

Explanation

Sets drawing environment parameters and executes the primitives registered in the OT.

The drawing environment specified by DrawOTagEnv() is effective until PutDrawEnv(), DrawOTagEnv() or the [DR_ENV](#) primitive are executed.

To detect when execution of the primitive list is complete, use DrawSync() or install a callback routine with DrawSyncCallback().

See also

[PutDrawEnv\(\)](#), [DrawOTagEnv\(\)](#), [DrawSync\(\)](#), [DrawSyncCallback\(\)](#)

DrawOTagIO

Draw the primitives registered in the OT

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
void DrawOTagIO(
u_long *p)           Pointer to top of OT
```

Explanation

Collectively executes the primitives registered in the OT. It is the same as DrawOTag(), except that it uses CPU I/O instead of DMA, which results in a significant speed decrease.

See also

[DrawOTag\(\)](#)

DrawPrim

Draw a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void DrawPrim(  
void *p)           Pointer to primitive
```

Explanation

Executes a primitive which has completed initialization. This routine blocks while waiting for all drawing commands in the queue to complete, then executes immediately.

See also

[DrawOTag\(\)](#)

DrawSync

Wait for all drawing to terminate.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

**long DrawSync(
long mode)**

0: Wait for termination of all non-blocking functions registered in the queue
1: Return the number of positions in the current queue

Explanation

Waits for drawing to terminate.

If DrawSync(0) is used, and execution of the primitive list takes an exceptionally long time (approximately longer than 8 Vsync) to complete, a timeout is generated and the GPU is reset. Reasons why this might occur include an exceptionally long primitive list, or one that renders exceptionally large numbers of pixels. Another possibility is that the primitive list has been corrupted in some way. To avoid this, the application can use a loop such as:

```
while(DrawSync(1));
```

The following routines use the GPU queue, and therefore their termination can be detected using DrawSync(), or by setting a callback with DrawSyncCallback(): ClearImage(), ClearImage2(), DrawOTag(), DrawOTagEnv(), LoadImage(), MoveImage(), PutDrawEnv(), StoreImage().

Return value

Number of positions in the execution queue.

See also

[DrawSyncCallback\(\)](#)

DrawSyncCallback

Define a callback function to be called when the GPU is finished executing a primitive list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/24/99

Syntax

```
u_long DrawSyncCallback(  
void (*func)())           Pointer to callback function
```

Explanation

Defines a routine to be used as a callback when drawing is completed. When all requests in the queue have terminated, the function *func* is called. If *func* is set to 0, then any previous callback routine is disabled.

Inside the callback, subsequent drawing termination interrupts are masked. Therefore, the callback routine should return as soon as possible. Although the specified function is called during an interrupt, it is not an interrupt handler; it should be written as a normal subroutine that is called by the main interrupt handler.

The following routines use the GPU queue, and therefore their termination can be detected using DrawSync(), or by setting a callback with DrawSyncCallback(): ClearImage(), ClearImage2(), DrawOTag(), DrawOTagEnv(), LoadImage(), MoveImage(), PutDrawEnv(), StoreImage().

It is important to note that the callback is called when the GPU queue is empty. If a particular set of drawing commands has terminated, but new commands have already been placed in the queue, the callback isn't called until all the commands have terminated.

Return Value

Pointer to the previously registered callback function.

See also

[DrawSync\(\)](#)

DumpClut, dumpClut

Print contents of *clut* member of primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void DumpClut(
u_short *clut*) CLUT ID

DumpClut(*clut*) Macro version of DumpClut().

Explanation

Prints the CLUT contents.

See also

[GetClut\(\)](#), [LoadClut\(\)](#)

DumpDispEnv

Print contents of display environment Structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void DumpDispEnv(  
    DISPENV *env)    Pointer to display environment
```

Explanation

Prints the contents of the display environment structure.

DumpDrawEnv

Print contents of drawing environment structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void DumpDrawEnv(  
DRAWENV *env)    Pointer to drawing environment
```

Explanation

Prints the contents of the drawing environment structure.

See also

[SetDrawEnv\(\)](#)

DumpOTag

Print primitives registered in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void DumpOTag(  
u_long *ot)           OT starting pointer
```

Explanation

Prints the code fields of the primitives registered in the OT.

See also

[DrawOTag\(\)](#)

DumpTPage, dumpTPage

Print contents of *tpage* member of primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void DumpTPage(
u_short *tpage*) Texture page ID

DumpTPage(*tpage*) Macro version of DumpTPage().

Explanation

Prints the contents of the texture page ID.

See also

[setTPage\(\)](#)

FntFlush

Draw contents of print stream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
u_long *FntFlush(  
long id)          Print stream ID
```

Explanation

Draws the contents of the print stream into the frame buffer. It initializes and then draws a sprite primitive list corresponding to the characters specified in the print stream.

When *id* is -1, the print stream ID which was set in SetDumpFnt() is used (0 if print stream ID was not set).

After the drawing has been done, the print stream contents are also flushed.

Return value

The starting pointer of the primitive list used to perform the drawing.

See also

[SetDumpFnt\(\)](#)

FntLoad

Transmit font pattern.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void FntLoad(
long tx, long ty)           Font pattern frame buffer address
```

Explanation

Transmits the built-in text font used for debugging text output to the frame buffer. It loads the basic font pattern (4-bit, 256x128) and initializes all the print streams.

FntLoad() must always be executed before FntOpen() and FntFlush(). The font area must not clash with the frame buffer area used by the application. Font data is located at the upper left of the texture page for FntFlush(). Font data is treated as a [RECT](#) (0,0,32,32) area consisting of 128 characters, each 128 x 32. As this is similar to the texture page area, *tx* is restricted to a multiple of 64 and *ty* is restricted to a multiple of 256.

Loads the Clut to location (*tx*, *ty*+128).

See also

[FntOpen\(\)](#), [FntFlush\(\)](#), [SetDumpFnt\(\)](#)

FntOpen

Open a print stream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/24/99

Syntax

```
long FntOpen(
long x, long y,           Display start location
long w, long h,           Display area
long isbg,                Automatic clearing of background
                                0: Don't clear background to (0, 0, 0) when display is performed
                                1: Clear background to (0, 0, 0) when display is performed
long n)                   Maximum number of characters
```

Explanation

Opens the stream for on-screen printing. After this, character strings up to *n* characters long can be drawn in the (*x*, *y*)- (*x*+*w*, *y*+*h*) rectangular area of the frame buffer, using `FntPrint()`. If *isbg* is 1, the background is cleared when a character string is drawn.

Up to 8 streams can be opened at once. However, once a stream is opened, it cannot be closed until the next time `FntLoad()` is called.

n specifies the maximum number of characters. Up to 1024 characters can be specified together in 8 streams.

Return value

The stream ID.

See also

[FntLoad\(\)](#), [FntPrint\(\)](#)

FntPrint

Print a string.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

long FntPrint(

long *id*,

char **format*)

Print stream ID

Pointer to print format

Explanation

Sends the string *format* to the specified print stream using the same interface as the `fprintf()` standard C library function.

The character string is not actually displayed until `FntFlush()` has been executed.

Return value

The number of characters in the stream.

See also

[FntOpen\(\)](#), [FntFlush\(\)](#)

GetClut, getClut

Calculate value of the *CLUT* member in a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

u_short GetClut(
int x, int y) Frame buffer address of CLUT

GetClut(*x, y*) Macro version of getClut().

Explanation

Calculates and returns the texture CLUT ID.

The CLUT address is limited to multiples of 16 in the x direction.

Return value

CLUT ID.

See also

[setClut\(\)](#)

GetDispEnv

Get current display environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

`DISPENV *GetDispEnv(
DISPENV *env)` Pointer to display environment start address

Explanation

Stores the current display environment in the address specified by *env*.

Return value

A pointer to the display environment obtained by the function.

See also

[PutDispEnv\(\)](#), [SetDefDispEnv\(\)](#)

GetDrawArea

Get data for the current draw area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

**void GetDrawArea(
[DR_AREA](#) *p)** Starting address for DR_AREA primitive

Explanation

Reads GPU's current draw area settings into *p*.

p must be initialized beforehand using SetDrawArea().

See also

[SetDrawArea\(\)](#)

GetDrawEnv

Get the current drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
DRAWENV *GetDrawEnv(  
DRAWENV *env)      Pointer to drawing environment start address
```

Explanation

Stores the current drawing environment in the address specified by *env*.

Return value

env starting address

See also

[PutDrawEnv\(\)](#), [SetDrawEnv\(\)](#), [GetDrawEnv2\(\)](#)

GetDrawEnv2

Get the current drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.6	9/1/99

Syntax

[DR_ENV](#) ***GetDrawEnv2**(
[DR_ENV](#) p) Pointer to drawing environment change primitive

Explanation

Gets the current drawing environment from the GPU and sets the drawing environment change primitive. The drawing environments that are obtained are as follows:

- DR_AREA (Drawing area)
- DR_OFFSET (Drawing offset)
- DR_TPAGE (Texture page)
- DR_TWIN (Texture window)
- DR_STP (STP bit processing)

Return value

None.

See also

[PutDrawEnv\(\)](#), [SetDrawEnv\(\)](#), [GetDrawEnv\(\)](#)

GetDrawMode

Get current draw mode data

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

**void GetDrawMode(
[DR_MODE](#) *p)** Starting address for DR_MODE primitives

Explanation

Reads GPU's current draw mode settings into *p*.

p must be initialized beforehand with SetDrawMode().

See also

[SetDrawMode\(\)](#)

GetDrawOffset

Get the current draw offset.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

**void GetDrawOffset(
[DR_OFFSET](#) *p)** Starting address for DR_OFFSET primitive

Explanation

Reads GPU's current draw offset settings into *p*.

p must be initialized beforehand with SetDrawOffset().

See also

[SetDrawOffset\(\)](#)

GetGraphDebug

Get present debug level.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

int GetGraphDebug(*void*)

Explanation

Gets graphics system debug level.

Return value

Present debug level value.

See also

GetODE

Get field currently being drawn.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

int GetODE(*void*)

Explanation

Gets field currently being drawn.

Return value

Current drawing field:

0: VRAM even address being drawn

1: VRAM odd address being drawn

GetTexWindow

Get current texture window data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

**void GetTexWindow(
[DR_TWIN](#) *p)** Starting address for DR_TWIN primitives

Explanation

Reads GPU's current texture window settings into *p*.

p must be initialized beforehand with SetTexWindow().

See also

[SetTexWindow\(\)](#)

GetTimSize

Calculate size of Tim data domain returned by Krom2Tim().

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
int GetTimSize(
u_char *sjis)           Pointer to sjis character string
```

Explanation

Calculates size of the Tim data domain returned by Krom2Tim(). This size domain is maintained in malloc() and is designated Krom2Tim().

Return value

Size of Tim data domain returned by Krom2Tim().

See also

[Krom2Tim\(\)](#)

GetTPage, getTPage

Calculate value of member *tpage* in a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

u_short GetTPage(

int *tp*, Texture mode
 0: 4bitCLUT
 1: 8bitCLUT
 2: 16bitDirect
int *abr*, Semitransparency rate
 0: 0.5 x Back + 0.5 x Forward
 1: 1.0 x Back + 1.0 x Forward
 2: 1.0 x Back - 1.0 x Forward
 3: 1.0 x Back + 0.25 x Forward
int *x*, int *y*) Texture page address

getTPage(*tp*, *abr*, *x*, *y*) Macro version of GetTPage().

Explanation

Calculates the texture page ID, and returns it.

The semitransparent rate is also effective for polygons on which texture mapping is not performed.

The texture page address is limited to a multiple of 64 in the X direction and a multiple of 256 in the Y direction.

Return value

Texture page ID.

See also

[setTPage\(\)](#), [DumpTPage\(\)](#)

IsEndPrim, isendprim

Determine if a primitive is the last in a list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

**int IsEndPrim(
void *p)**

Primitive start address

isendprim(p)

Macro version of IsEndPrim().

Explanation

Decides if the end of the primitive list is *p*.

Return value

1: final end case; 0: non-final end case.

See also

[AddPrim\(\)](#)

IsIdleGPU

Check if drawing suspended by BreakDraw() was completed.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.6	12/14/98

Syntax

```
int IsIdleGPU(  
int maxcount)
```

Count value

Explanation

Checks whether the GPU is idle.

When drawing is suspended by BreakDraw(), the GPU doesn't stop until drawing of the current primitive is completed. This function checks whether the drawing suspended by BreakDraw() has completed. *maxcount* is the number of times the function will check for idle before returning.

Return value

0: GPU is in idle state. -1: GPU is in drawing state.

See also

[BreakDraw\(\)](#)

KanjiFntClose

Close print streams.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

void KanjiFntClose(void)

Explanation

Closes all the streams currently open and are by KanjiFntPrint() and initializes the state of the Kanji font routines. It will function correctly even when no streams are open.

See also

[KanjiFntFlush\(\)](#), [KanjiFntOpen\(\)](#), [KanjiFntPrint\(\)](#)

KanjiFntFlush

Draw contents of a Kanji print stream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
u_long *KanjiFntFlush(
int id)                Print stream ID
```

Explanation

Draws the contents of the Kanji print stream into the frame buffer. It initializes and then draws a sprite primitive list corresponding to the characters specified in the print stream.

The contents of a print stream are also flushed after the end of drawing.

To internally reserve the transfer buffer on the stack, approximately 72K is needed.

Return value

Start pointer of a primitive list used for drawing

See also

[KanjiFntClose\(\)](#), [KanjiFntOpen\(\)](#), [KanjiFntPrint\(\)](#)

KanjiFntOpen

Open a print stream.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	2/24/99

Syntax

int KanjiFntOpen(

int x, int y,

int w, int h,

int dx, int dy,

int cx, int cy,

int isbg,

Position of starting display

Display area

Kanji font pattern frame buffer address

Kanji clut frame buffer address

Automatic background clear

0: Does not clear the background to (0, 0, 0) during display

1: Clears the background to (0, 0, 0) during display

int n)

Maximum number of characters

Explanation

Opens a stream for screen printing. Then, KanjiFntPrint() can be used to render a character string composed of up to *n* characters in the rectangular area of (*x*, *y*) and (*x*+*w*, *y*+*h*) in the frame buffer. With *isbg* assigned a value of one, the background is cleared when a character string is rendered.

Up to four streams can be opened at a time, and a maximum of 256 characters can be specified in one stream. The kanji font area must not interfere in the frame buffer area used for applications.

With *dx* and *dy*, the address in the upper left of the texture page must be specified.

Return value

Stream ID.

See also

[KanjiFntClose\(\)](#), [KanjiFntFlush\(\)](#), [KanjiFntPrint\(\)](#)

KanjiFntPrint

Print a string, in SJIS ZENKAKU format.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
int KanjiFntPrint(
int id,           Print stream ID
char *format, [arg]...)  Pointer to print format
```

Explanation

Send SJIS ZENKAKU string using printf() interface.

KANJI code must be the SJIS. Although both ZENKAKU and HANKAKU characters can be mixed in the string, a HANKAKU character is converted to ZENKAKU when it is drawn. HANKAKU KANA characters are not supported. Actual drawing of the string is done at execution of KanjiFntFlush(). When there is ~p in the string format, all the characters after ~p are drawn in half-pitch.

Return value

Number of characters within the stream.

See also

[KanjiFntClose\(\)](#), [KanjiFntFlush\(\)](#), [KanjiFntOpen\(\)](#)

Krom2Tim

Convert SJIS character string to 4-bit CLUT Tim data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
int Krom2Tim (
u_char *sjis,          SJIS character string
u_long *taddr,         Tim area for storing data
int dx, int dy,        Pixel data x,y coordinates on VRAM
int cdx, int cdy,      Clut data x,y coordinates on VRAM
u_int fg, u_int bg)    Character color and bg color
```

Explanation

Converts SJIS character string to 4 bits CLUT TIM data and returns it to *taddr*.

The size area returned by GetTimSize() must be secured in advance.

The Kanji code must be SJIS. Full-width and half-width characters can be mixed within the character string, but when they are displayed, they area ll converted to full-width characters. Half-width characters are not supported.

To internally reserve the transfer buffer on the stack, approximately 72K is needed.

Return value

When an abnormal code is given, -1 is returned; 0 otherwise.

See also

[GetTimSize\(\)](#)

LoadClut

Load 256-color CLUT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

u_short LoadClut(

u_long **clut*, Pointer to CLUT data start address

long *x*, **long** *y*) Destination coordinates in frame buffer

Explanation

Loads 256 entries of texture color data (CLUT) from main memory address *clut* into the frame buffer (*x*,*y*) and calculates the ID of the loaded texture CLUT.

256 palette entries are always transmitted, even in 4-bit mode.

Return value

The CLUT ID for the loaded CLUT.

See also

[LoadClut2\(\)](#)

LoadClut2

Load 16-color CLUT.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
u_short LoadClut2(
u_long *clut,      Texture color start address
int x, int y)      Destination frame buffer address
```

Explanation

Loads 16 entries of texture color data (CLUT) from main memory address *clut* into the frame buffer (x,y) and calculates the ID of the loaded texture CLUT.

LoadClut2() transmits 16 palette entries whereas LoadClut() transmits 256 palette entries.

LoadClut2() internally invokes LoadImage().

Return value

The CLUT ID for the loaded CLUT.

See also

[LoadClut\(\)](#), [LoadImage\(\)](#)

LoadImage

Transfer data to a frame buffer.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

int LoadImage()

[RECT](#) *recp,

Pointer to destination rectangular area

u_long *p)

Pointer to main memory address of source of transmission

Explanation

Transfers the contents of memory from the address *p* to the rectangular area in the frame buffer specified by *recp*.

Because LoadImage() is a non-blocking function, transmission termination must be detected by DrawSync() or by installing a callback routine with DrawSyncCallback().

The source and destination areas are not affected by the drawing environment (clip, offset). The destination area must be located within a drawable area (0, 0) - (1023, 511). See the description of the [DR_LOAD](#) primitive.

Return value

Position of this command in the libgpu command queue.

See also

[DrawSync\(\)](#), [DrawSyncCallback\(\)](#), [LoadImage2\(\)](#), [StoreImage\(\)](#),

LoadImage2

Transfer data to a frame buffer (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	12/14/98

Syntax

int LoadImage2(

RECT *rect,

Pointer to destination rectangular area

u_long *p)

Pointer to main memory address of source of transfer

Explanation

Transfers the contents of memory beginning at the address pointed to by *p* to the rectangular area in the frame buffer specified by *rect*, without queueing. This operation takes place immediately, regardless of what is in the GPU queue, and on completion, processing of the queue is resumed.

When drawing has been suspended using `BreakDraw()` and you want to transfer data to the frame buffer using `LoadImage()`, immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use `LoadImage2()`.

The drawing area (clip offset) does not affect the transfer area.

The destination area must be located within a drawable area (0, 0) - (1023, 511).

When drawing is suspended with `BreakDraw()` after `LoadImage2()` is called, before restarting the drawing with `ContinueDraw()`, it is necessary to confirm the completion of data transfer using `IsIdleGPU()`. This is because `LoadImage2()` is a non-blocking function.

Return value

0: Normal completion; -1: Abnormal completion.

See also

[BreakDraw\(\)](#), [ContinueDraw\(\)](#), [IsIdleGPU\(\)](#), [LoadImage\(\)](#)

LoadTPage

Load a texture page.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

u_short LoadTPage(

u_long **pix*,

int *tp*,

int *abr*,

int *x*, **int** *y*,

int *w*, **int** *h*)

Pointer to texture pattern start address

Bit depth (0 = 4-bit; 1 = 8-bit; 2 = 16-bit)

Semitransparency rate

Destination frame buffer address

Texture pattern size

Explanation

Loads a texture pattern from the memory area starting at the address *pix* into the frame buffer area starting at the address (*x*, *y*), and calculates the texture page ID for the loaded texture pattern.

The texture pattern size *w* represents the number of pixels, not the actual size of the transfer area in the frame buffer.

LoadTPage() calls LoadImage() internally.

Return value

Texture page ID for the loaded texture pattern.

See also

[LoadImage\(\)](#), [setTPage\(\)](#)

MargePrim

Link primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.0	12/14/98

Syntax

```
int MargePrim(
void *p0,      First primitive
void *p1)     Second primitive
```

Explanation

Links primitive *p0* to primitive *p1*. The combined primitive size of *p0* and *p1* must be less than 15 words. Within this size, any number of connections is possible.

The resulting linked primitives can be added to an OT using `AddPrim()`.

p0 and *p1* describe continuous regions of memory. *p1* must be the higher address.

Return value

0 on success, -1 on failure.

See also

[AddPrim\(\)](#)

MovelImage

Transfer data between two locations within the frame buffer.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

int MovelImage(

RECT *rect,

int x, int y)

Pointer to source rectangular area

Top left corner of the destination rectangle

Explanation

The rectangular area of the frame buffer specified by *rect* is copied to the rectangular area of the same size starting at (x, y).

The content at the source is preserved. If the source and destination areas are the same, normal operation is not guaranteed.

Because MovelImage() is a non-blocking function, the end of copying must be detected by DrawSync() or by installing a callback routine with DrawSyncCallback().

Note: Depending on the timing, there are cases when MovelImage() fails to execute when multiple MovelImage() functions are executed while a movie is playing. In such cases, it is necessary to wait for the transfer to terminate by calling DrawSync() after each MovelImage().

The source and destination areas are not affected by the drawing environment (clip, offset). The destination area must be located within a drawable area (0, 0) - (1023, 511). See also the description of the DR_MOVE primitive.

Return value

Position of this command in the libgpu command queue.

See also

[DrawSync\(\)](#), [DrawSyncCallback\(\)](#), [LoadImage\(\)](#), [MovelImage2\(\)](#)

MovelImage2

Transfer data between two locations within the frame buffer (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	12/14/98

Syntax

int MovelImage2(

[RECT](#) **rect*,

int *x*, *y*)

Pointer to source rectangular area

Top left corner of the destination rectangle

Explanation

The rectangular area of the frame buffer specified by *rect* is transferred to the rectangular area of the same size starting at (*x*, *y*). This operation takes place immediately, regardless of what is in the GPU queue, and on completion, processing of the queue is resumed.

The contents of the source rectangle are preserved. If the source and destination areas are the same, normal operation is not guaranteed.

When drawing is suspended with `BreakDraw()` and you want to move data within the frame buffer using `MovelImage()`, immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use `MovelImage2()`.

The source and destination transfer areas are not affected by the drawing environment (clip, offset). The source and destination areas must be located within a drawable area (0, 0) - (1023, 511). See also the description of the [DR_MOVE](#) primitive.

When drawing is suspended with `BreakDraw()` after `MovelImage2()` is called, before restarting the drawing with `ContinueDraw()`, it is necessary to confirm the completion of data transfer using `IsIdleGPU()`. This is because `MovelImage2()` is a non-blocking function.

Return value

0: Normal completion; -1: Abnormal completion..

See also

[BreakDraw\(\)](#), [ContinueDraw\(\)](#), [IsIdleGPU\(\)](#), [MovelImage\(\)](#)

NextPrim, nextPrim

Get pointer to next primitive in primitive list.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

**void *NextPrim(
void *p)** Pointer to start address of a primitive

nextPrim(p) Macro version of NextPrim()

Explanation

Returns a pointer to the next primitive in a primitive list.

Return value

Pointer to the next primitive.

See also

[AddPrim\(\)](#)

OpenTIM

Open TIM data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

**long OpenTIM(
u_long *addr)** Pointer to main memory address to which the TIM has been loaded

Explanation

Opens a TIM in main memory. The information in the opened TIM can then be read using ReadTIM().

Only one TIM can be opened at a time. An opened TIM is not closed until the next time OpenTIM() is called.

Return value

0 on success; any other value indicates failure.

See also

[ReadTIM\(\)](#)

OpenTMD

Open TMD data.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

long OpenTMD(
u_long *tmd,
long obj_no)

Pointer to main memory address to which TMD has been loaded
Object number

Explanation

Opens the TMD of the object specified by *obj_no*. The information in the opened TMD can then be read using ReadTMD().

Calling OpenTMD() closes any previously opened TMD.

Return value

The number of polygons comprising the object as a positive integer; on failure, returns 0.

See also

[ReadTMD\(\)](#)

PutDispEnv

Set the display environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

[DISPENV](#) *PutDispEnv(
[DISPENV](#) *env) Pointer to display environment start address

Explanation

Sets a display environment according to information specified by *env*.

Return value

A pointer to the display environment set; on failure, returns 0.

See also

[GetDispEnv\(\)](#), [SetDefDispEnv\(\)](#), [SetDefDispEnv\(\)](#)

PutDrawEnv

Set the drawing environment.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
DRAWENV *PutDrawEnv(  
DRAWENV *env)      Pointer to drawing environment start address
```

Explanation

The basic drawing parameters (such as the drawing offset and the drawing clip area) are set according to the values specified in *env*.

The drawing environment is effective until the next time PutDrawEnv() is executed, or until the [DR_ENV](#) primitive is executed.

Return value

A pointer to the drawing environment set. On failure, returns 0.

See also

[DrawOTagEnv\(\)](#), [GetDrawEnv\(\)](#), [SetDefDrawEnv\(\)](#), [SetDrawEnv\(\)](#)

ReadTIM

Produce TIM header.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
TIM_IMAGE *ReadTIM(
TIM_IMAGE *timing)    TIM_IMAGE structure pointer
```

Explanation

Sets the members of the [TIM_IMAGE](#) structure pointed to by *timing* according to the data specified by the most recent OpenTIM() call.

Return value

The *timing* start address, if succesful; 0 if TIM analysis fails.

See also

OpenTim()

ReadTMD

Read contents of TMD primitives.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	2/24/99

Syntax

TMD_PRIM *ReadTMD(
TMD_PRIM *tmdprim) Pointer to printer for TMD-PRIM structure

Explanation

Sets the members of the **TMD_PRIM** structure pointed to by *tmdprim* according to the data specified by the most recent **OpenTMD()** call.

Note that the **TMD_PRIM** structure includes fields that are not used for all types of objects. **ReadTIM()** copies only those fields that are valid for the current object.

Gradation system primitives are not supported.

Return value

tmdprim if successful; 0 on failure.

See also

OpenTMD()

ResetGraph

Initialize drawing engine.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
int ResetGraph(  
int mode)           Reset mode
```

Explanation

Resets the graphic system according to *mode*:

Table 7-3

Mode	Operation
0	Complete reset. The drawing environment and display environment are initialized.
1	Cancels the current drawing and flushes the command buffer.
3	Initializes the drawing engine while preserving the current display environment (i.e. the screen is not cleared or the screen mode changed).

SetDefDispEnv

Set display environment structure members and screen display area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

DISPENV

*SetDefDispEnv(

DISPENV *disp,

int x, int y,

int w, int h)

Pointer to display environment

Upper left corner of display area

Width and height of the display area

Explanation

Sets the members of a DISPENV (display environment) structure. The new display area is specified using the coordinates within the frame buffer of the top left corner, along with the width and height, of the desired rectangle.

Table 7-4

Member	Content	Value
disp	Display area	(x, y, w, h)
screen	Screen display area	(0, 0)-(0, 0)
isinter	Interlace flag	0
isrgb24	24-bit mode flag	0

This function does not actually change the display environment. It merely sets the members of the specified structure as desired. Use PutDispEnv() with this structure to change the actual environment.

Note: While the *screen* width and height are set to (0, 0), internally they are processed as (256, 240).

Return value

Pointer to the display environment set.

See also

[GetDispEnv\(\)](#), [PutDispEnv\(\)](#)

SetDefDrawEnv

Set standard drawing environment structure.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
DRAWENV *SetDefDrawEnv(  
DRAWENV *env,           Pointer to drawing environment  
int x, int y,           Upper left corner of drawing area  
int w, int h)           Width and height of drawing area
```

Explanation

Sets the drawing area members of a DRAWENV (drawing environment) structure. The new drawing area is specified using the coordinates within the frame buffer of the top left corner, along with the width and height, of the desired rectangle.

Table 7-5

Member	Content	Value
clip	Drawing area	(x, y, w, h)
ofs[2]	Drawing offset	(x, y)
tw	Texture window	(0, 0, 0, 0)
tpage	Texture page (tp, abr, tx, ty)	(0, 0, 640, 0)
dtd	Dither processing flag	1 (ON)
dfe	Permission flag for drawing	1 (drawing on display area is inhibited)
isbg	Draw area clear flag	0 (clear: OFF)
r0, g0, b0	Background color	(0, 0, 0)

This function does not actually change the drawing environment. It merely sets the members of the specified structure as desired. Use PutDrawEnv() with this structure to change the actual environment.

Return value

The starting pointer of the drawing environment set.

See also

[GetDrawEnv\(\)](#), [PutDrawEnv\(\)](#), [SetDrawEnv\(\)](#)

SetDispMask

Set and cancel display mask.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void SetDispMask(  
int mask)           Display mask
```

Explanation

Puts display mask into the status specified by *mask*. *mask* = 0: not displayed on screen; *mask* = 1; displayed on screen.

SetDrawArea

Initialize content of drawing area setting primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetDrawArea(

[DR_AREA](#) *p,

[RECT](#) *r)

Pointer to drawing area setting primitive

Pointer to drawing area

Explanation

Initializes a DR_AREA primitive. By using AddPrim() to insert a DR_AREA primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

See also

[AddPrim\(\)](#), [GetDrawArea\(\)](#)

SetDrawEnv

Initialize content of drawing environment change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetDrawEnv(

DR_ENV **dr_env*,

DRAWENV **env*)

Pointer to drawing environment change primitive

Pointer to drawing environment structure in which the drawing environment is described

Explanation

Initializes a DR_ENV primitive using the values contained in a DRAWENV structure. By using AddPrim() to insert a DR_ENV primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

The DR_ENV primitive uses the same information as the DRAWENV structure, but the data format is different and the DRAWENV structure cannot be used as a primitive. When the DR_ENV primitive is executed, the previous drawing environment settings are destroyed.

See also

AddPrim(), GetDrawEnv(), PutDrawEnv(), SetDefDrawEnv()

SetDrawLoad

Initialize content of the LoadImage primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void SetDrawLoad(
  DR_LOAD *p,           Destination rectangular area primitive
  RECT *rect)           Destination rectangular area
```

Explanation

Initializes the destination rectangular area primitive. By registering the initialized primitive in OT using AddPrim(), the rectangular area can be transferred just as in LoadImage().

Maximum data transfer amount is 12 words (24 pixels).

See also

[AddPrim\(\)](#), [LoadImage\(\)](#)

SetDrawMode, setDrawMode

Initialize content of a drawing mode primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	9/1/99

Syntax

void SetDrawMode(

DR_MODE *p,

int dfe,

int dtd,

int tpage,

RECT *tw)

Pointer to drawing mode primitive

0: drawing not allowed in display area,

1: drawing allowed in display area

0: dithering off, 1: dithering on.

Texture page

Pointer to texture window

setDrawMode (p, dfe, dtd, tpage, tw) Macro version of SetDrawMode()

Explanation

Initializes a DR_MODE primitive. By using AddPrim() to insert a DR_MODE primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

If *tw* is 0, the texture window is not changed.

See also

[GetDrawMode\(\)](#)

SetDrawMove

Initialize rectangle copy primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetDrawMove(

DR_MOVE *p,

RECT *rect,

int x,y)

Pointer to rectangular area copy primitive

Rectangular area to be transferred

Upper left edge of the rectangular area transfer destination

Explanation

Initializes the rectangular area copy primitive. After the primitive is initialized, it can be entered in the OT using AddPrim(). When the primitive is executed, it performs the same copying of a rectangular area as MoveImage().

See also

AddPrim(), MoveImage()

SetDrawOffset

Initialize drawing offset setting primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetDrawOffset(

DR_OFFSET *p, Pointer to drawing offset setting primitive

u_short *ofs) Pointer to drawing offset

Explanation

Initializes a DR_OFFSET primitive. By using AddPrim() to insert a DR_OFFSET primitive into your primitive list, it is possible to change part of your drawing environment in the middle of drawing.

See also

[AddPrim\(\)](#), [GetDrawOffset\(\)](#)

SetDrawStp

Initializes STP bit update primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	12/14/98

Syntax

void SetDrawStp(

DR_STP *p,

int pbw)

Pointer to primitive

STP bit update flag (0: STP bit OFF, 1: STP bit ON)

Explanation

Initializes the DR_STP primitive pointed to by p.

When *pbw* = 0, normal drawing is performed. When *pbw* = 1, drawing is performed with the STP bit set (STP is a 16-bit object).

SetDrawTPage, setDrawTPage

Initialize texture page change primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetDrawTPage(

DR_TPAGE *p,

int dfe,

int dtd,

int tpage)

Texture page setting primitive

Flag for drawing to a display area

0: no drawing in display area

1: drawing in display area

Dither processing flag

0: dither processing not performed

1: dither processing performed

Texture page

setDrawTPage(p, dfe, dtd, tpage)

Macro version of SetDrawTPage().

Explanation

Initializes a texture page change primitive. By registering the initialized primitive in OT using AddPrim(), the texture page can be changed while drawing.

See also

AddPrim(), setTPage()

SetDumpFnt

Define stream for onscreen dump.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void SetDumpFnt(
long id)           Print stream ID
```

Explanation

Sets the print stream for debug printing. The output of the debug printing functions can then be carried out in relation to the stream specified in *id*.

The actual display is executed by FntFlush().

See also

[dumpRECT\(\)](#), [dumpMatrix\(\)](#), [dumpVector\(\)](#), [dump...\(\)](#), [FntFlush\(\)](#)

SetGraphDebug

Set debugging level.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void SetGraphDebug(
int level)           Debugging level
```

Explanation

Sets a debugging level for the graphics system. *level* can be one of the following:

Table 7-6

Level	Operation
0	No checks are performed. (Highest speed mode)
1	Checks coordinating registered and drawn primitives.
2	Registered and drawn primitives are dumped.

Return value

The previously set debug level.

See also

[GetGraphDebug\(\)](#)

**SetLineF2, SetLineF3, SetLineF4;
setLineF2, setLineF3, setLineF4;
SetLineG2, SetLineG3, SetLineG4;
setLineG2, setLineG3, setLineG4**

Initialize a line primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetLineF2 (LINE_F2 *p)	Flat unconnected straight line drawing primitive.
void SetLineF3 (LINE_F3 *p)	Flat connected 2-straight line drawing primitive.
void SetLineF4 (LINE_F4 *p)	Flat connected 3-straight line drawing primitive.
void SetLineG2 (LINE_G2 *p)	Gouraud unconnected straight line drawing primitive.
void SetLineG3 (LINE_G3 *p)	Gouraud connected 2-straight line drawing primitive.
void SetLineG4 (LINE_G4 *p)	Gouraud connected 3-straight line drawing primitive.

setLineF2 (p)	Macro version of SetLineF2()
setLineF3 (p)	Macro version of SetLineF3()
setLineF4 (p)	Macro version of SetLineF4().
setLineG2 (p)	Macro version of SetLineG2()
setLineG3 (p)	Macro version of SetLineG3()
setLineG4 (p)	Macro version of SetLineG4()

Explanation

These functions initialize the primitives specified by *p*.

See also

SetPolyF3, SetPolyF4;
setPolyF3, setPolyF4;
SetPolyG3, SetPolyG4;
setPolyG3, setPolyG4;
SetPolyGT3, SetPolyGT4;
setPolyGT3, setPolyGT4

Initialize a polygon primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetPolyF3 (POLY_F3 *p)	Flat shaded triangle primitive.
void SetPolyF4 (POLY_F4 *p)	Flat shaded quadrangle primitive.
void SetPolyFT3 (POLY_FT3 *p)	Flat textured triangle primitive.
void SetPolyFT4 (POLY_FT4 *p)	Flat textured quadrangle primitive.
void SetPolyG3 (POLY_G3 *p)	Gouraud shaded triangle primitive.
void SetPolyG4 (POLY_G4 *p)	Gouraud shaded quadrangle primitive.
void SetPolyGT3 (POLY_GT3 *p)	Gouraud textured triangle primitive.
void SetPolyGT4 (POLY_GT4 *p)	Gouraud textured quadrangle primitive.

setPolyF3 (p)	Macro version of SetPolyF3()
setPolyF4 (p)	Macro version of SetPolyF4()
setPolyFT3 (p)	Macro version of SetPolyFT3()
setPolyFT4 (p)	Macro version of SetPolyFT4()
setPolyG3 (p)	Macro version of SetPolyG3()
setPolyG4 (p)	Macro version of SetPolyG4()
setPolyGT3 (p)	Macro version of SetPolyGT3()
setPolyGT4 (p)	Macro version of SetPolyGT4()

Explanation

These functions initialize the primitive specified by *p*.

See also

SetSemiTrans, setSemiTrans

Set the semitransparent attribute of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetSemiTrans(

void **p*,

int *abe*)

Pointer to primitive

Semitransparent flag

0: semitransparent OFF

1: semitransparent ON

setSemiTrans(*p*, *abe*) Macro version of SetSemiTrans()

Explanation

Sets the semitransparent attribute of the primitive specified by *p* to the value specified by *abe*. If semitransparent mode is enabled, then semitransparent pixels are drawn as specified by the table below.

Table 7-7

Primitive	Pixels subjected to semitransparent processing
POLY_FT3/POLY_FT4	Pixels for which the topmost bit of the corresponding texture pixel is 1
POLY_GT3/POLY_GT4	Pixels for which the topmost bit of the corresponding texture pixel is 1
SPRT/SPRT_8/SPRT_16	Pixels for which the topmost bit of the corresponding texture pixel is 1
Other drawing primitives	All Pixels

Semitransparent pixels are calculated from the foreground pixels *Pf* and background pixels *Pb* as follows:

$$P = F \times Pf + B \times Pb$$

The rate (F, B) of semitransparency is designated by the member *tpage* in the primitive. Drawing speed is reduced because semitransparency requires reading of background brightness values. Therefore, do not draw primitives with semitransparent mode turned on unless they are to be displayed that way.

SetShadeTex, setShadeTex

Inhibit shading function.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax**void SetShadeTex(****void **p*,****int *tge*)**

Pointer to primitive

Unshaded flag

0: Shading is performed

1: Shading is not performed

setShadeTex(*p*, *tge*)

Macro version of SetShadeTex()

ExplanationSets the shading attribute of the primitive pointed to by *p* to the value specified by *tge*.

When texture and shading are both ON, each pixel in the polygon is calculated as shown below from the pixel value *T* of the corresponding texture pattern, and the brightness value *L* corresponding to the pixel value *T*.

```

P = (T&0xf8*L&0xf8) / 128
if (p > 255)      p = 255
if (p < 0)        p = 0

```

When *L* = 128, the brightness value of the texture pattern is drawn as it is. If the value results in an overflow, the pixel value is clipped to 255.

When *tge* = 1, the brightness value is not divided, and the texture pattern value is used, as it is, as the pixel value.

T, *L* are only effective for the upper 5 bits. The lower 3 bits are discarded.

This function cannot be used for primitives other than POLY_FT3, POLY_FT4, SPRT, SPRT_8, and SPRT_16.

Although the texture number of colors is saved at intensity level of 32 when using ShadeTex, the shading brightness value is decreased from an intensity level of 256 to 32.

SetSprt, SetSprt8, SetSprt16; setSprt, setSprt8, setSprt16

Initialize a sprite primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetSprt(SPRT **p*) Initialize a SPRT primitive.
void SetSprt8(SPRT_8 **p*) Initialize a SPRT8 primitive.
void SetSprt16(SPRT_16 **p*) Initialize a SPRT16 primitive.

setSprt(*p*) Macro version of SetSprt()
setSprt8(*p*) Macro version of SetSprt8()
setSprt16(*p*) Macro version of SetSprt16()

Explanation

These functions initialize the primitives specified by *p*. Details are given below.

Table 7-8

Function name	Sprite size	Primitive
SetSprt8	8 x 8	SPRT_8
SetSprt16	16 x 16	SPRT_16
SetSprt	Can be set using values of members h, w. (0 < h < 255, 0 < w < 255)	SPRT

The SPRT... primitives are faster than [POLY_FT4](#). [TILE](#) is also faster than [POLY_F4](#).

SetTexWindow

Initialize the content of a texture window primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

```
void SetTexWindow(
  DR_TWIN *p,           Pointer to texture window primitive
  RECT *tw)             Pointer to texture window
```

setTexWindow(*p*, *tw*) Macro version of SetTexWindow()

Explanation

Initializes a DR_TWIN primitive using the specified values. By using AddPrim() to insert a DR_TWIN primitive into your primitive list, it is possible to change the current texture window in the middle of drawing.

See also

[AddPrim\(\)](#), [GetTexWindow\(\)](#)

SetTile, SetTile1, SetTile8, SetTile16; setTile, setTile1, setTile8, setTile16

Initialize a tile primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

void SetTile(TILE *p) Initialize a TILE primitive.
void SetTile1(TILE_1 *p) Initialize a TILE1 primitive.
void SetTile8(TILE_8 *p) Initialize a TILE8 primitive.
void SetTile16(TILE_16 *p) Initialize a TILE16 primitive.

setTile(p) Macro version of SetTile()
setTile1(p) Macro version of SetTile1()
setTile8(p) Macro version of SetTile8()
setTile16(p) Macro version of SetTile16()

Explanation

These functions initialize the primitives specified by *p*. Details are given below.

Table 7-9

Function name	Tile size	Primitive size
SetTile1	1 x 1	TILE_1
SetTile8	8 x 8	TILE_8
SetTile16	16 x 16	TILE_16
SetTile	Can be set using values of members h, w. (0 < h < 255, 0 < w < 255)	TILE

The [SPRT](#) primitives are faster than [POLY_FT4](#). TILE is also faster than POLY_F4.

StoreImage

Transfer image data from the frame buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

int StoreImage(

RECT *recp,

Pointer to destination rectangular area

u_long *p)

Pointer to main memory address of destination of transmission

Explanation

Transfers the rectangular area of the frame buffer specified by *recp* to the main memory storage location starting at the address specified by *p*.

Because StoreImage() is a non-blocking function, use DrawSync() to determine when the operation has completed, or install a callback with DrawSyncCallback().

The source and destination areas are not affected by the drawing environment (clip, offset). The source area must be located within a drawable area (0, 0) - (1023, 511).

Return value

Position of this command in the libgpu command queue.

See also

[StoreImage2\(\)](#), [DrawSync\(\)](#), [DrawSyncCallback\(\)](#), [LoadImage\(\)](#)

StoreImage2

Transfer data from a frame buffer (immediate execution).

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.1	12/14/98

Syntax

int StoreImage2(

RECT *rect,

Pointer to source rectangular area

u_long *p)

Pointer to destination main memory address

Explanation

Transfers the contents of the rectangular area in the frame buffer pointed to by *rect* to the main memory location starting with the address pointed to by *p*, without queueing. *p* must be on a word boundary. This operation takes place immediately, regardless of what is in the GPU queue, and on completion, processing of the queue is resumed.

When drawing is suspended with `BreakDraw()`, and you want to transfer data from the frame buffer using `StoreImage()`, immediate execution is not possible because of the need for queueing. If immediate execution is desired, you must use `StoreImage2()`.

The drawing area (clip offset) does not affect the transfer area.

The transfer area must be located within a drawable area (0, 0) - (1023, 511).

When drawing is suspended with `BreakDraw()` after `StoreImage2()` is called, before restarting the drawing with `ContinueDraw()`, it is necessary to confirm the completion of data transfer using `IsIdleGPU()`. This is because `StoreImage2()` is a non-blocking function.

Return value

0: Normal completion. -1: Abnormal completion.

See also

[BreakDraw\(\)](#), [ContinueDraw\(\)](#), [IsIdleGPU\(\)](#), [StoreImage\(\)](#), [LoadImage\(\)](#)

TermPrim, termPrim

Terminate a primitive list

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

**void TermPrim(
void *p)** Pointer to start address of a primitive list

termPrim(p) Macro version of TermPrim()

Explanation

Sets the tag pointer of the primitive specified by *p* to point at a special terminator value that signals the end of the list when it is executed. Any primitives already pointed to by *p* are removed from the list.

See also

[CatPrim\(\)](#), [MargePrim\(\)](#)

VSync

Wait for the next vertical blank, or return the vertical blank counter value.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	12/14/98

Syntax

```
int VSync(  
int mode)
```

Mode

Explanation

Waits for vertical blank using the method specified by *mode*, as defined below.

Table 7-10

Mode	Operation
0	Blocks until vertical sync is generated
1	Returns time elapsed from the point VSync() processing is last completed
n (n>1)	Blocks from the point VSync() processing is last completed when mode=1 or n in horizontal sync units
-n (n>0)	Returns absolute time after program boot in vertical sync interval units.

Vsync() may generate a timeout if long blocking periods are specified. To prevent deadlocks, rather than using Vsync() to block for an especially long time (say more than 4 vertical blank periods), have your program poll VSync(-1) in a loop instead.

Return value

Mode value is as listed below.

Table 7-11

Mode	Return value
mode>=0	Time elapsed from the point that Vsync() processing is last completed when mode=1 or n (horizontal blanking units)
mode<0	Time elapsed after program boot (vertical blanking units)

See also

[DrawSync\(\)](#), [VSynCallback\(\)](#)

VSyncCallback

Define a function to be executed during each vertical blank period.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	12/14/98

Syntax

```
void VSyncCallback(
void (*func)())      Pointer to callback function
```

Explanation

Specifies that the routine at address *func* should be executed at the start of the vertical blank interrupt. If *func* is 0, then any previous callback routine is disabled.

Subsequent interrupts are masked inside *func*. Therefore, it is necessary to return quickly after performing necessary processes using *func*.

Although the specified function is called during an interrupt, it is not the actual interrupt handler. It should be written as a normal subroutine that are called by the main interrupt handler.

See also

[VSync\(\)](#), [DrawSyncCallback\(\)](#)

Macros

addVector

Add vectors.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

addVector(*v0*, *v1*)

Arguments

v0, *v1* Pointers to vectors

Explanation

Adds *v1* to the vector *v0*, and stores the result in *v0*.

Since it is a macro, there is no dependence on the vector type.

See also

[applyVector\(\)](#), [copyVector\(\)](#)

applyVector

Apply arithmetic operation to a vector.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	3.4	12/14/98

Syntax

applyVector(*v, x, y, z, op)

Arguments

<i>v</i>	Pointer to vector
<i>x,y,z</i>	Coordinate value
<i>op</i>	Operator

Explanation

Performing the operation specified on vector *v*, *x*, *y*, *z* and *op*

```
applyVector (v, 2, 4, 8, +=)
```

is equivalent to:

```
v->vx += 2, v-> vy += 4, v-> vz += 8
```

applyVector is a macro, so there is no dependence on the vector model.

See also

[addVector\(\)](#), [copyVector\(\)](#)

copyVector

Copy vectors.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

copyVector(*v0*, *v1*)

Arguments

v0, *v1* Vector pointer

Explanation

Copies vector *v1* to *v0*.

`copyVector` is a macro, so there is no dependence on the vector type.

See also

[addVector\(\)](#), [applyVector\(\)](#)

dumpMatrix

Display matrix contents.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

dumpMatrix(x)

Arguments

x pointer to matrix

Explanation

Displays the contents of the matrix pointed to by x.

If SetDumpFmt() is called, the output is displayed to the screen. Otherwise, the output is sent to stdout.

See also

[SetDumpFmt\(\)](#), [dumpRECT\(\)](#), [dumpVector\(\)](#), [dump...](#)()

dumpRECT

Display contents of a rectangle.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

dumpRECT(*r*)

Arguments

r pointer to RECT

Explanation

Displays the contents of the [RECT](#) structure pointed to by *r*.

If SetDumpFnt() is called, the output is displayed to the screen. Otherwise, the output is sent to stdout.

Since this is a macro, it does not depend on the vector type.

See also

[SetDumpFnt\(\)](#), [dumpMatrix\(\)](#), [dumpVector\(\)](#), [dump...](#)

dumpVector

Display contents of vector

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax**dumpVector**(*str*, *v*)**Arguments**

str string to be displayed
v pointer to vector

Explanation

This macro displays the contents of the vector pointed to by *v*.

If `SetDumpFnt()` is called, the output is displayed to the screen. Otherwise, the output is sent to `stdout`.

See also

[SetDumpFnt\(\)](#), [dumpRECT\(\)](#), [dumpVector\(\)](#), [dump...](#)()

dump...

Display contents of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

dumpWH(<i>p</i>)	Displays w, h of primitive.
dumpXY0(<i>p</i>)	Displays x0, y0 of primitive.
dumpXY2(<i>p</i>)	Displays x0, x1, y0, y1 of primitive.
dumpXY3(<i>p</i>)	Displays x0-x2, y0-y2 of primitive.
dumpXY4(<i>p</i>)	Displays x0-x3, y0-y3 of primitive.
dumpUV0(<i>p</i>)	Displays u0, v0 of primitive.
dumpUV3(<i>p</i>)	Displays u0-u2, v0-v2 of primitive.
dumpUV3(<i>p</i>)	Displays u0-u3, v0-v3 of primitive.
dumpRGB0(<i>p</i>)	Displays r0, g0, b0 of primitive.
dumpRGB1(<i>p</i>)	Displays r0, r1, g0, g1, b0, b1 of primitive.
dumpRGB2(<i>p</i>)	Displays r0-r2, g0-g2, b0-b2 of primitive.
dumpRGB3(<i>p</i>)	Displays r0-r3, g0-g3, b0-b3 of primitive.

Explanation

Displays the contents of the primitive pointed to by p. If SetDumpFnt() is called, the output is displayed to the screen. Otherwise, the output is sent to stdout.

Since these are macros, they do not depend on the primitive type.

See also

[SetDumpFnt\(\)](#)

setClut

Set CLUT for primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

setClut(*p*, *x*, *y*)

Arguments

<i>p</i>	pointer to primitive
<i>x</i> , <i>y</i>	CLUT frame buffer address

Explanation

The texture CLUT ID specified by the parameters is set in the clut member of primitive *p*.

Since this is a macro, it does not depend on the primitive type.

For the CLUT address, the *x* component must be a multiple of 16.

See also

[DumpClut\(\)](#), [GetClut\(\)](#), [LoadClut\(\)](#)

setRECT

Set rectangular area.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setRECT(*r*, *x*, *y*, *w*, *h*)

Arguments

<i>r</i>	Pointer to RECT structure
<i>x</i> , <i>y</i>	Upper left point of rectangular area
<i>w</i> , <i>h</i>	Size of rectangular area

Explanation

This macro sets the *x*, *y*, *w*, and *h* values of the [RECT](#) structure *r*.

See also

[dumpRECT\(\)](#)

setRGB0, setRGB1, setRGB2, setRGB3

Initialize RGB fields of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setRGB0(*p*, *r0*, *g0*, *b0*) Initialize *r0*, *g0*, and *b0* fields
setRGB1(*p*, *r1*, *g1*, *b1*) Initialize *r1*, *g1*, and *b1* fields
setRGB2(*p*, *r2*, *g2*, *b2*) Initialize *r2*, *g2*, and *b2* fields
setRGB3(*p*, *r3*, *g3*, *b3*) Initialize *r3*, *g3*, and *b3* fields

Arguments

p Pointer to primitive
r, *g*, *b* RGB members of primitive

Explanation

These macros set the values for the RGB members of the primitive *p*.

These are macros, so there is no dependence on the primitive type.

setTPage

Set texture page for primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

setTPage(*p*, *tp*, *abr*, *x*, *y*)

Arguments

<i>p</i>	pointer to primitive
<i>tp</i>	texture mode 0: 4bitCLUT 1: 8bitCLUT 2: 16bitDirect
<i>abr</i>	semi-transparency rate 0: 0.5 x Back + 0.5 x Forward 1: 1.0 x Back + 1.0 x Forward 2: 1.0 x Back - 1.0 x Forward 3: 1.0 x Back + 0.25 x Forward
<i>x</i> , <i>y</i>	texture page address

Explanation

The texture page specified by the parameters is set in the *tpage* member of primitive *p*.
Since this is a macro, it does not depend on the primitive type.

See also

[GetTPage\(\)](#), [SetDrawTPage\(\)](#), [LoadTPage\(\)](#)

setUV0, setUV3, setUV4

Set the *u* and *v* members of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setUV0 (* <i>p</i> , <i>u0</i> , <i>v0</i>)	Set <i>u0</i> and <i>v0</i>
setUV3 (* <i>p</i> , <i>u0</i> , <i>v0</i> , <i>u1</i> , <i>v1</i> , <i>u2</i> , <i>v2</i>)	Set <i>u0-u2</i> and <i>v0-v2</i>
setUV4 (* <i>p</i> , <i>u0</i> , <i>v0</i> , <i>u1</i> , <i>v1</i> , <i>u2</i> , <i>v2</i> , <i>u3</i> , <i>v3</i>)	Set <i>u0-u3</i> and <i>v0-v3</i>

Arguments

<i>p</i>	Pointer to primitive
<i>u</i> , <i>v</i>	UV members of primitive

Explanation

These macros set the values of the appropriate UV fields of the primitive *p*.

These are C preprocessor macros and can be used with any primitive or structure with the appropriate fields.

See also

[setUVWH\(\)](#)

setUVWH

Set the *u*, *v* members of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setUVWH(**p*, *u0*, *v0*, *w*, *h*)

Arguments

<i>p</i>	Pointer to primitive
<i>u0</i> , <i>v0</i>	Upper left corner of primitive texture
<i>w</i> , <i>h</i>	Width and height of primitive texture

Explanation

This macro sets the *u0*, *v0*, *u1*, *v1*, *u2*, *v2*, *u3*, and *v3* fields of a primitive structure to represent the corners of the rectangle specified by the input parameters.

It can be used with any primitive or structure with the appropriate fields; it cannot be used with sprite primitives.

See also

[setUV0\(\)](#)

setVector

Set the values of a vector.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setVector(*v, x, y, z)

Arguments

v	Pointer to a vector
x, y, z	Coordinate values

Explanation

Sets the (x, y, z) values of a VECTOR or SVECTOR (defined in libgte).

setWH

Set the *w,h* members of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setWH(**p*, *w*, *h*)

Arguments

p Pointer to primitive.
w, h Width and height of primitive texture

Explanation

Specifies the *w,h* members of a primitive (cannot be used with primitives which do not have *w,h* members).

See also

[setXYWH\(\)](#)

setXY0, setXY2, setXY3, setXY4

Set the x and y parameters of a primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	2.x	12/14/98

Syntax

setXY0(*p, x0, y0)

setXY2(*p, x0, y0, x1, y1)

setXY3(*p, x0, y0, x1, y1, x2, y2)

setXY4(*p, x0, y0, x1, y1, x2, y2, x3, y3)

Arguments

p

Pointer to primitive

x, y

x, y members of primitive

Explanation

These macros set the values for the *x, y* members of the primitive. Because they are macros, there is no dependence on the primitive type.

See also

[setXYWH\(\)](#)

setXYWH

Set x , y members of a rectangular primitive.

Library	Header File	Introduced	Documentation Date
<i>libgpu.lib</i>	<i>libgpu.h</i>	4.3	12/14/98

Syntax

setXYWH(p , $x0$, $y0$, w , h)

Arguments

p	pointer to primitive
$x0$, $y0$	upper left point of primitive
w , h	width and height of primitive

Explanation

The coordinates of the rectangle defined by $(x0, y0)$ - $(x0 + w, y0 + h)$ are set in members $(x0, y0)$.. $(x3, y3)$ of the primitive. This macro works with any primitive having $(x0, y0)$.. $(x3, y3)$ members.

See also

[setXY0\(\)](#), [setWH\(\)](#)

Chapter 8: Basic Geometry Library

Table of Contents

Structures

CRVECTOR3	8-5
CRVECTOR4	8-6
CVECTOR	8-7
DIVPOLYGON3	8-8
DIVPOLYGON4	8-9
DVECTOR	8-10
EVECTOR	8-11
MATRIX	8-12
POL3	8-13
POL4	8-14
QMESH	8-15
RVECTOR	8-16
SPOL	8-17
SVECTOR	8-18
TMESH	8-19
VECTOR	8-20

Functions

ApplyMatrix	8-21
ApplyMatrixLV	8-22
ApplyMatrixSV	8-23
ApplyRotMatrix	8-24
ApplyRotMatrixLV	8-25
ApplyTransposeMatrixLV	8-26
AverageZ3	8-27
AverageZ4	8-28
catan	8-29
ccos	8-30
Clip3F, Clip3FP, Clip3FT, Clip3FTP, Clip3G, Clip3GT, Clip3GTP	8-31
Clip4F, Clip4FP, Clip4FT, Clip4FTP, Clip4G, Clip4GT, Clip4GTP	8-33
cln	8-35
ColorCol	8-36
ColorDpq	8-37
ColorMatCol	8-38
ColorMatDpq	8-39
CompMatrix	8-40
CompMatrixLV	8-41
csin	8-42
csqrt	8-43
DivideF3, DivideF4, DivideFT3, DivideFT4, DivideG3, DivideG4, DivideGT3, DivideGT4	8-44
DpqColor	8-46
DpqColor3	8-47
DpqColorLight	8-48
EigenMatrix	8-49
gteMIMefunc	8-50
InitClip	8-51
InitGeom	8-52
Intpl	8-53
InvSquareRoot	8-54
IsIdMatrix	8-55
LightColor	8-56
LoadAverage0	8-57
LoadAverage12	8-58
LoadAverageByte	8-59

LoadAverageCol	8-60
LoadAverageShort0	8-61
LoadAverageShort12	8-62
LocalLight	8-63
Lzc	8-64
MatrixNormal	8-65
MatrixNormal_0	8-66
MatrixNormal_1	8-67
MatrixNormal_2	8-68
MulMatrix	8-69
MulMatrix0	8-70
MulMatrix2	8-71
MulRotMatrix	8-72
MulRotMatrix0	8-73
NormalClip	8-74
NormalColor, NormalColor_nom	8-75
NormalColor3, NormalColor3_nom	8-76
NormalColorCol, NormalColorCol_nom	8-77
NormalColorCol3, NormalColorCol3_nom	8-78
NormalColorDpq, NormalColorDpq_nom	8-79
NormalColorDpq3, NormalColorDpq3_nom	8-80
otz2p	8-81
OuterProduct0	8-82
OuterProduct12	8-83
p2otz	8-84
pers_map	8-85
PhongLine	8-86
PopMatrix	8-87
PushMatrix	8-88
ratan2	8-89
rcos	8-90
RCpolyF3, RCpolyFT3, RCpolyG3, RCpolyGT3	8-91
RCpolyF4, RCpolyFT4, RCpolyG4, RCpolyGT4	8-92
ReadColorMatrix	8-93
ReadGeomOffset	8-94
ReadGeomScreen	8-95
ReadLightMatrix	8-96
ReadRGBfifo	8-97
ReadRotMatrix	8-98
ReadSXSIfifo	8-99
ReadSZfifo3	8-100
ReadSZfifo4	8-101
RotAverage3, RotAverage3_nom	8-102
RotAverage4	8-103
RotAverageNclip3	8-104
RotAverageNclip3_nom	8-105
RotAverageNclip4	8-106
RotAverageNclipColorCol3	8-107
RotAverageNclipColorCol3_nom	8-108
RotAverageNclipColorDpq3	8-109
RotAverageNclipColorDpq3_nom	8-110
RotColorDpq	8-111
RotColorDpq_nom	8-112
RotColorDpq3	8-113
RotColorDpq3_nom	8-114
RotColorMatDpq	8-115
RotMatrix...	8-116

RotMatrix_gte	8-118
RotMatrixC	8-119
RotMatrixX	8-120
RotMatrixY	8-121
RotMatrixYZ_gte	8-122
RotMatrixZ	8-123
RotMatrixZYX_gte	8-124
RotMeshH	8-125
RotMeshPrimQ_T	8-126
RotMeshPrimR_...	8-127
RotMeshPrimS_...	8-128
RotNclip3	8-129
RotNclip3_nom	8-130
RotNclip4	8-131
RotPMD_...	8-132
RotPMD_SV_...	8-133
RotRMD_...	8-134
RotRMD_SV_...	8-135
RotSMD_...	8-136
RotSMD_SV_...	8-137
RotTrans	8-138
RotTrans_nom	8-139
RotTransPers	8-140
RotTransPers_nom	8-141
RotTransPers3	8-142
RotTransPers3_nom	8-143
RotTransPers3N	8-144
RotTransPers4	8-145
RotTransPers4_nom	8-146
RotTransPersN	8-147
RotTransSV	8-148
rsin	8-149
ScaleMatrix	8-150
ScaleMatrixL	8-151
SetBackColor	8-152
SetColorMatrix	8-153
SetFarColor	8-154
SetFogFar	8-155
SetFogNear	8-156
SetFogNearFar	8-157
SetGeomOffset	8-158
SetGeomScreen	8-159
SetLightMatrix	8-160
SetMulMatrix	8-161
SetMulRotMatrix	8-162
SetRGBcd	8-163
SetRotMatrix	8-164
SetTransMatrix	8-165
Square0	8-166
Square12	8-167
SquareRoot0	8-168
SquareRoot12	8-169
SquareSL0	8-170
SquareSL12	8-171
SquareSS0	8-172
SquareSS12	8-173
SubPol3	8-174

SubPol4	8-175
TransMatrix	8-176
TransposeMatrix	8-177
TransRotPers	8-178
TransRotPers3	8-179
TransRot_32	8-180
VectorNormal	8-181
VectorNormalS	8-182
VectorNormalSS	8-183

CRVECTOR3

Triangular recursive vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {  
    RVECTOR r01, r12, r20;    Division vertex vector data  
    RVECTOR *r0, *r1, *r2;    Pointer to division vector data  
    u_long *rtn;              Pointer to return address for assembler  
} CRVECTOR3;
```

See also

[RCpolyF3\(\)](#)

CRVECTOR4

Quadrilateral recursive vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    RVECTOR r01, r02, r31, r32, rc;
    RVECTOR *r0, *r1, *r2, *r3;
    u_long *rtn;
} CRVECTOR4;
```

Division vertex vector data
Pointer to division vertex vector data
Pointer to return address for assembler

See also

[RCpolyF4\(\)](#)

CVECTOR

Character vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
typedef struct {  
    u_char r, g, b;  
    u_char cd;  
};
```

Color palette
GPU code

DIVPOLYGON3

Triangular division buffer.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    u_long ndiv;           Number of divisions
    u_long pih, piv;       Clip area specification (display screen resolution)
    u_short clut;          CLUT
    u_short tpage;         Texture page
    CVECTOR rgbc;          Code + RGB color
    u_long *ot;            Pointer to OT
    RVECTOR r0, r1, r2;    Division vertex vector data
    CRVECTOR3 cr[5];       Triangular recursive vector data
} DIVPOLYGON3;
```

See also

[DivideF3\(\)](#), [RCpolyF3\(\)](#)

DIVPOLYGON4

Quadrilateral recursive vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {  
    u_long ndiv;           Number of divisions  
    u_long pih, piv;       Clip area specification (display screen's resolution)  
    u_short clut;          CLUT  
    u_short tpage;         Texture page  
    CVECTOR rgbc;          Code + RGB color  
    u_long *ot;            Pointer to OT  
    RVECTOR r0, r1, r2, r3; Division vertex vector data  
    CRVECTOR4 cr[5];       Quadrilateral recursive vector data  
} DIVPOLYGON4;
```

See also

[DivideF4\(\)](#), [RCpolyF4\(\)](#)

DVECTOR

2D vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {  
    short vx, vy;           Vector coordinates  
} DVECTOR;
```

See also

[RotMeshH\(\)](#), [RotTransPers3N\(\)](#), [RotTransPersN\(\)](#)

EVECTOR

Clip vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    SVECTOR v;           Local object 3D vertex
    VECTOR xyz;          Screen 3D vertex
    DVECTOR sxy;         Screen 2D vertex
    CVECTOR rgb;         Color palette
    short txuv, pad;     Texture mapping data
    long chx, chy;       Clip area data
} EVECTOR;
```

See also

[Clip3F\(\)](#), [Clip4F\(\)](#), [InitClip\(\)](#)

MATRIX

Matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct MATRIX {
    short m[3][3];           3 x 3 matrix coefficient value
    long t[3];              Parallel transfer volume
};
```

Explanation

Specifies each component on the MATRIX $m[i][j]$. Specifies the transfer volume after conversion on the MATRIX $t[i]$. Pay attention to the differing word lengths on m and t .

The GTE essentially performs the following multiply and accumulate calculations from the MATRIX structure.

1. RotTrans system function (function group which does not perform coordinate conversion). Performs only basic matrix calculations and vector addition.

```
MATRIX      m
SVECTOR     xi
SVECTOR     xo
[ xo.vx ]   [ m.m[0][0] m.m[0][1] m.m[0][2] ] [ xi.vx ] [ m.t[0] ]
[ xo.vy ] = [ m.m[1][0] m.m[1][1] m.m[1][2] ] [ xi.vy ] [ m.t[1] ]
[ xo.vz ]   [ m.m[2][0] m.m[2][1] m.m[2][2] ] [ xi.vz ] [ m.t[2] ]
```

2. RotTransPers system function (function group which performs coordinate conversion). In addition to the (a) calculation, perspective conversion (division by z) is performed at the same time.

```
MATRIX      m
SVECTOR     xi
SVECTOR     xo
SVECTOR     x2
long        h
[ xo.vx ]   [ m.m[0][0] m.m[0][1] m.m[0][2] ] [ xi.vx ]   [ m.t[0] ]
[ xo.vy ] = [ m.m[1][0] m.m[1][1] m.m[1][2] ] [ xi.vy ] + [ m.t[1] ]
[ xo.vz ]   [ m.m[2][0] m.m[2][1] m.m[2][2] ] [ xi.vz ]   [ m.t[2] ]
```

$$x2.vx = (h * xo.vx) / xo.vz$$

$$x2.vy = (h * xo.vy) / xo.vz$$

POL3

Triangle polygon.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct POL3 {
    short sxy[3][2];      Screen coordinates
    short sz[3][2];       Screen coordinates
    short uv[3][2];       Texture coordinates
    short rgb[3][3];      RGB value
    short code;           Code (F3 = 1, FT3 = 2, G3 = 3, GT3 = 4)
};
```

See also

[SubPol3\(\)](#)

POL4

Four-sided polygon.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct POL4 {
    short sxy[4][2];      Screen coordinates
    short sz[4][2];       Screen coordinates
    short uv[4][2];       Texture coordinates
    short rgb[4][3];      RGB value
    short code;           Code (F4 = 5, FT4 = 6, G4 = 7, GT4 = 8)
};
```

See also

[SubPol4\(\)](#)

QMESH

Quadrilateral mesh

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.3	12/14/98

Structure

```
typedef struct {
    SVECTOR *v;      pointer to shared vertex coordinates array
    SVECTOR *n;      pointer to shared normal array
    SVECTOR *u;      pointer to shared texture coordinates array
    CVECTOR *c;      pointer to shared color data array
    u_long lenv;     vertex length (horizontal)
    u_long lenh;     vertex length (vertical)
}QMESH;
```

See also

[RotMeshPrimQ_T\(\)](#)

RVECTOR

Division vertex vector data.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Structure

```
typedef struct {  
    SVECTOR v;           Local object 3D vertex  
    u_char uv[2];        Texture mapping data  
    u_short pad;  
    CVECTOR c;           Vertex color palette  
    DVECTOR sxy;         Screen 2D vertex  
    u_long sz;           Clip Z-data  
} RVECTOR;
```

See also

[RCpolyF3\(\)](#), [RCpolyF4\(\)](#)

SPOL

Vertex information.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct SPOL {
    short xy[3];           XY coordinates
    short uv[2];           UV coordinates
    short rgb[3];          RGB value
};
```

See also

[SubPol3\(\)](#), [SubPol4\(\)](#)

SVECTOR

Short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct SVECTOR {  
    short vx, vy, vz;  
    short pad;  
};
```

Vector coordinates
System reserved

TMESH

Triangle mesh.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct TMESH {
    SVECTOR *v;           Pointer to vertex string
    SVECTOR *n;           Pointer to normal string
    SVECTOR *u;           Pointer to texture string
    CVECTOR *c;           Pointer to RGB string
    u_long len;           Mesh length
};
```

See also

[RotMeshPrimR...\(\)](#), [RotMeshPrimS...\(\)](#)

VECTOR

Vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Structure

```
struct VECTOR {  
    long vx, vy, vz;           Vector coordinates  
    long pad;                  System reserved  
};
```

ApplyMatrix

Multiply a vector by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
VECTOR *ApplyMatrix(
MATRIX *m,           Pointer to matrix to be multiplied (input)
SVECTOR *v0,         Pointer to short vector (input)
VECTOR *v1)          Pointer to vector (output)
```

Explanation

Multiplies the matrix *m* by the short vector *v0* beginning with the rightmost end. The vector is in effect rotated and then translated.

The result is saved in the vector *v1*. The function destroys the constant rotation matrix.

```
m -> m [[i] [j]]      : (1, 3, 12)
v0 -> vx, vy, vz      : (1, 15, 0)
v1 -> vx, vy, vz      : (1, 31, 0)
```

Return value

v1.

See also

[ApplyMatrixLV\(\)](#), [ApplyMatrixSV\(\)](#), [ApplyRotMatrix\(\)](#), [ApplyRotMatrixLV\(\)](#), [ApplyTransposeMatrixLV\(\)](#)

ApplyMatrixLV

Multiply a vector by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

VECTOR ***ApplyMatrixLV**(
 MATRIX **m*, Pointer to matrix to be multiplied (input)
 VECTOR **v0*, Pointer to vector (input)
 VECTOR **v1*) Pointer to vector (output)

Explanation

Multiplies matrix *m* by vector *v0* beginning from the rightmost end. The result is saved in vector *v1*. It is a 16 x 32 bit multiplier which uses the GTE. It destroys the constant rotation matrix.

m -> *m* [i] [j] : (1, 3, 12)
v0 -> *vx*, *vy*, *vz* : (1, 31, 0)
v1 -> *vx*, *vy*, *vz* : (1, 31, 0)

Return value

v1

See also

[ApplyMatrix\(\)](#), [ApplyMatrixSV\(\)](#), [ApplyRotMatrix\(\)](#), [ApplyRotMatrixLV\(\)](#), [ApplyTransposeMatrixLV\(\)](#)

ApplyMatrixSV

Multiply a vector by a matrix.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	3.0	12/14/98

Syntax

SVECTOR *ApplyMatrixSV(
MATRIX *m,
SVECTOR *v0,
SVECTOR *v1)

Pointer to matrix to be multiplied (input)
Pointer to short vector (input)
Pointer to short vector (output)

Explanation

Multiplies matrix *m* by short vector *v0* beginning at the rightmost end. The result is saved in the short vector *v1*. This function destroys the rotation matrix.

m -> m [i] [j]
v0 -> vx, vy, vz
v1 -> vx, vy, vz

: (1, 3, 12)
: (1, 15, 0)
: (1, 15, 0)

Return value

v1

See also

[ApplyMatrix\(\)](#), [ApplyMatrixLV\(\)](#), [ApplyRotMatrix\(\)](#), [ApplyRotMatrixLV\(\)](#), [ApplyTransposeMatrixLV\(\)](#)

ApplyRotMatrix

Multiply a vector by a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	3.0	12/14/98

Syntax

```
VECTOR *ApplyRotMatrix(  
SVECTOR *v0,           Pointer to short vector (input)  
VECTOR *v1)            Pointer to vector (output)
```

Explanation

Multiplies a constant rotation matrix by short vector *v0* beginning at the rightmost end. The result is saved in vector *v1*.

```
v0 -> vx, vy, vz      : (1, 15, 0)  
v1 -> vx, vy, vz      : (1, 31, 0)
```

Return value

v1

See also

[ApplyMatrix\(\)](#), [ApplyMatrixLV\(\)](#), [ApplyMatrixSV\(\)](#), [ApplyRotMatrixLV\(\)](#), [ApplyTransposeMatrixLV\(\)](#)

ApplyRotMatrixLV

Multiply a vector by a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

VECTOR ***ApplyRotMatrixLV**(
VECTOR **v0*, Pointer to long vector (input)
VECTOR **v1*) Pointer to vector (output)

Explanation

Multiplies a constant rotation matrix by long vector *v0* beginning at the rightmost end. The result is saved in vector *v1*.

v0 -> vx, vy, vz : (1, 31, 0)
v1 -> vx, vy, vz : (1, 31, 0)

Return value

v1

See also

[ApplyMatrix\(\)](#), [ApplyMatrixLV\(\)](#), [ApplyMatrixSV\(\)](#), [ApplyRotMatrix\(\)](#), [ApplyTransposeMatrixLV\(\)](#)

ApplyTransposeMatrixLV

Multiply a vector by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

VECTOR *ApplyTransposeMatrixLV(
 MATRIX *m, Pointer to matrix to be multiplied
 VECTOR *v0, Pointer to vector (input)
 VECTOR *v1) Pointer to vector (output)

Explanation

Multiplies a transposed matrix by vector *v0* beginning at the rightmost end. The result is saved in vector *v1*.

m -> m [i] [j] : (1, 3, 12)
v0 -> vx, vy, vz : (1, 31, 0)
v1 -> vx, vy, vz : (1, 31, 0)

Return value

v1

See also

[ApplyMatrix\(\)](#), [ApplyMatrixLV\(\)](#), [ApplyMatrixSV\(\)](#), [ApplyRotMatrix\(\)](#), [ApplyRotMatrixLV\(\)](#)

AverageZ3

Average three values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

**long AverageZ3(
long *sz0*, long *sz1*, long *sz2*)** Input values

Explanation

Calculates an average of three values *sz0*, *sz1*, and *sz2*.

sz0, *sz1*, *sz2* : (0, 16, 0)

Return value : (0, 16, 0)

Return value

Average of 1/4 of three values *sz0*, *sz1*, and *sz2*.

See also

[AverageZ4\(\)](#)

AverageZ4

Average four values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

long AverageZ4(
long sz0, long sz1, long sz2, long sz3) Input values

Explanation

Calculates an average of four values sz0, sz1, sz2, and sz3.

sz0, sz1, sz2, sz3 : (0, 16, 0)

Return value : (0, 16, 0)

Return value

1/4 of the average of four values sz0, sz1, sz2, and sz3.

See also

[AverageZ3\(\)](#)

catan

Compute arctangent of an angle within 180 degrees.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
int catan(  
int a)          Value
```

Explanation

Uses PlayStation® format (where 4096 = 360 degrees = 2 π) to find the arctan (between -90 and +90 degrees, $-\pi/2 \dots \pi/2$) of a .

a : (1, 19, 12)

Return value

atan (a) : (1, 19, 12)

See also

[ratan2\(\)](#), [ccos\(\)](#), [csin\(\)](#)

CCOS

Compute cosine of an angle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

**int ccos(
int a)** Angle (in PlayStation format)

Explanation

Finds the cosine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2 \text{ pi}$) using fixed point math (where $4096 = 1.0$).

Compared to `rcos()`, `ccos()` is slower and takes up less space

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\text{pi}$)

Return value

`cos (a)` : (1, 19, 12)

See also

[rcos\(\)](#), [csin\(\)](#), [catan\(\)](#)

Clip3F, Clip3FP, Clip3FT, Clip3FTP, Clip3G, Clip3GT, Clip3GTP

Three-vertex clipping functions.

Syntax

Flat-shaded, no perspective transformation.

long Clip3F(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **EVECTOR** **evmx)

Flat-shaded, with perspective transformation

long Clip3FP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **EVECTOR** **evmx)

Flat-shaded, textured, no perspective transformation

long Clip3FT(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **short** *uv0, **short** *uv1, **short** *uv2, **EVECTOR** **evmx)

Flat-shaded, textured, with perspective transformation

long Clip3FTP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **short** *uv0, **short** *uv1, **short** *uv2, **EVECTOR** **evmx)

Gouraud-shaded, no perspective transformation

long Clip3G(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **EVECTOR** **evmx)

Gouraud-shaded, with perspective transformation

long Clip3GP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **EVECTOR** **evmx)

Gouraud-shaded, textured, no perspective transformation

long Clip3GT(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **short** *uv0, **short** *uv1, **short** *uv2, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **EVECTOR** **evmx;

Gouraud-shaded, textured, with perspective transformation

long Clip3GTP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **short** *uv0, **short** *uv1, **short** *uv2, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **EVECTOR** **evmx;

Explanation

These functions clip a triangle having vertices v0, v1, and v2 to six surfaces defined by InitClip(). Other input parameters are:

uv0, uv1, uv2: Pointers to texture coordinate vectors
rgb0, rgb1, rgb2: Pointers to vertex color data

Effective output clip vector data is stored in evmx. The following members of evmx are returned depending on the particular function:

evmx[i] -> v	Local Object 3D Vertex (all functions)
evmx[i] -> xyz	Screen 3D Vertex (all functions)
evmx[i] -> xyz.pad	Fog effect interpolation value (P functions)
evmx[i] -> sxy	Screen 2D Vertex (P functions)
evmx[i] -> rgb	Vertex Color Data (G functions)
evmx[i] -> txuv	Texture Mapping Data (T functions)
evmx[i] -> chx	chx = vz * (hw/2)/h (all functions)
evmx[i] -> chy	chy = vz * (vw/2)/h (all functions)

These functions reserve the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

Output number of vertices.

See also

[Clip4F\(\)](#), [InitClip\(\)](#)

Clip4F, Clip4FP, Clip4FT, Clip4FTP, Clip4G, Clip4GT, Clip4GTP

Four-vertex clipping functions.

Syntax

Flat-shaded, no perspective transformation.

long Clip4F(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **EVECTOR****evmx)

Flat-shaded, with perspective transformation

long Clip4FP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **EVECTOR** **evmx)

Flat-shaded, textured, no perspective transformation

long Clip4FT(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **short** *uv0, **short** *uv1, **short** *uv2, **short** *uv3, **EVECTOR** **evmx)

Flat-shaded, textured, with perspective transformation

long Clip4FTP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **short** *uv0, **short** *uv1, **short** *uv2, **short** *uv3, **EVECTOR** **evmx)

Gouraud-shaded, no perspective transformation

long Clip4G(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **EVECTOR** **evmx)

Gouraud-shaded, with perspective transformation

long Clip4GP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **CVECTOR** *rgb3, **EVECTOR** **evmx)

Gouraud-shaded, textured, no perspective transformation

long Clip4GT(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **short** *uv0, **short** *uv1, **short** *uv2, **short** *uv3, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **CVECTOR** *rgb3, **EVECTOR** **evmx;

Gouraud-shaded, textured, with perspective transformation

long Clip4GTP(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3, **short** *uv0, **short** *uv1, **short** *uv2, **short** *uv3, **CVECTOR** *rgb0, **CVECTOR** *rgb1, **CVECTOR** *rgb2, **CVECTOR** *rgb3, **EVECTOR** **evmx;

Explanation

These functions clip a quadrilateral (linked triangle) having vertices v0, v1, v2, and v3 to six surfaces defined by InitClip(). Other input parameters are:

uv0, uv1, uv2, uv3:

Pointers to texture coordinate vectors

rgb0, rgb1, rgb2, rgb3:

Pointers to vertex color data

Effective output clip vector data is stored in evmx. The following members of evmx are returned depending on the particular function:

evmx[i] -> v

Local Object 3D Vertex (all functions)

evmx[i] -> xyz

Screen 3D Vertex (all functions)

evmx[i] -> xyz.pad

Fog effect interpolation value (P functions)

evmx[i] -> sxy

Screen 2D Vertex (P functions)

evmx[i] -> rgb

Vertex Color Data (G functions)

evmx[i] -> txuv

Texture Mapping Data (T functions)

evmx[i] -> chx

chx = vz * (hw/2)/h (all functions)

evmx[i] -> chy

chy = vz * (vw/2)/h (all functions)

These functions reserve the pointer arrays (20 pointer arrays = 80 bytes), including the work area.

Return value

Output number of vertices.

See also

[Clip3F\(\)](#), [InitClip\(\)](#)

cln

C logarithm function.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
int cln(
int a)           Value
```

Explanation

Uses fixed point math (where 4096 = 1.0) to find the fixed point natural logarithm.

a : (1, 19, 12)

Return value

ln (*a*) : (1, 19, 12)

ColorCol

Find a local color from a local light vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void ColorCol(  
VECTOR *v0,           Pointer to local light vector (input)  
CVECTOR *v1,          Pointer to primary color vector (input)  
CVECTOR *v2)          Pointer to color vector (output)
```

Explanation

Calculates the following:

```
LC = BK + LCM * v0  
v2 = v1 * LC (product of multiplication)  
v0 -> vx, vy, vz      : (1, 19, 12)  
v1 -> r, g, b          :(0, 8, 0)  
v2 -> r, g, b          :(0, 8, 0)
```

See also

[ColorDpq\(\)](#), [ColorMatCol\(\)](#), [ColorMatDpq\(\)](#)

ColorDpq

Find a local color from a local light vector, and perform depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void ColorDpq(
  VECTOR *v0,           Pointer to local light vector (input)
  CVECTOR *v1,          Pointer to primary color vector (input)
  long p,               Interpolation value for depth cueing (input)
  CVECTOR *v2)          Pointer to color vector (output)
```

Explanation

Calculates the following:

$$LC = BK + LCM \times v0$$

$$v2 = (1-p) \times v1 \times LC + p \times FC$$

where $v1 \times LC$ is the product of separate multiplication.

```
v0 -> vx, vy, vz      : (1, 19, 12)
v1 -> r, g, b         : (0, 8, 0)
p                     : (0, 20, 12)
v2 -> r, g, b         : (0, 8, 0)
```

See also

[ColorCol\(\)](#), [ColorMatCol\(\)](#), [ColorMatDpq\(\)](#)

ColorMatCol

Find a color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void ColorMatCol(  
  SVECTOR *v0,           Pointer to normal vector (input)  
  CVECTOR *v1,           Pointer to primary color vector (input)  
  CVECTOR *v2,           Pointer to color vector (output)  
  long matc)             Material (output)
```

Explanation

Performs the following calculations:

```
LLV = LLM * v0  
LLV = LLV^ (2^matc)  
LC = BK + LCM * LLV  
v2 = v1 * LC (separate multiplications)  
v0 -> vx, vy, vz      : (1, 3, 12)  
v1 -> r, g, b         : (0, 8, 0)  
v2 -> r, g, b         : (0, 8, 0)  
matc                   : (0, 32, 0)
```

See also

[ColorCol\(\)](#), [ColorDpq\(\)](#), [ColorMatDpq\(\)](#)

ColorMatDpq

Find a color and perform depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void ColorMatDpq(
 SVECTOR *v0, Pointer to normal vector (input)
 CVECTOR *v1, Pointer to color vector (input)
 long p, Interpolation value for depth cueing (output)
 CVECTOR *v2, Pointer to color vector (output)
 long matc) Material (output)

Explanation

Performs the following calculations:

$LLV = LLM \times v0$
 $LLV = LLV^{(2^{matc})}$
 $LC = BK + LCM \times LLV$
 $v2 = (1-p) \times v1 \times LC + p \times FC.$

v0 -> vx, vy, vz : (1, 3, 12)
v1 -> r, g, b : (0, 8, 0)
p : (0, 20, 12)
v2 -> r, g, b : (0, 8, 0)
matc : (0, 32, 0)

See also

[ColorCol\(\)](#), [ColorDpq\(\)](#), [ColorMatCol\(\)](#)

CompMatrix

Make a composite coordinate transformation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
MATRIX *CompMatrix(  
  MATRIX *m0, MATRIX *m1,    Pointer to matrix (input)  
  MATRIX *m2)                Pointer to matrix (output)
```

Explanation

Makes a composite coordinate transformation matrix that includes parallel translation.

$$[m2 \rightarrow m] = [m0 \rightarrow m] * [m1 \rightarrow m]$$
$$(m2 \rightarrow t) = [m0 \rightarrow m] * (m1 \rightarrow t) + (m0 \rightarrow t)$$

However, the values of the elements of $m1 \rightarrow t$ should be in the range $(-2^{15}, 2^{15})$.

$m0 \rightarrow m [i] [j]$: (1, 3, 12)
$m0 \rightarrow t [i]$: (1, 31, 0)
$m1 \rightarrow m [i] [j]$: (1, 3, 12)
$m1 \rightarrow t [i]$: (1, 15, 0)
$m2 \rightarrow m [i] [j]$: (1, 3, 12)
$m2 \rightarrow t [i]$: (1, 31, 0)

This function destroys a constant rotation matrix.

Return value

m2

See also

[CompMatrixLV\(\)](#)

CompMatrixLV

Make a composite coordinate transformation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.6	12/14/98

Syntax

```
MATRIX *CompMatrix(  
MATRIX *m0, MATRIX *m1,    Pointer to matrix (input)  
MATRIX *m2)                Pointer to matrix (output)
```

Explanation

Makes a composite coordinate transformation matrix that includes parallel translation.

```
[m2->m] = [m0->m] * [m1->m]  
(m2->t) = [m0->m] * (m1->t) + (m0->t)  
m0 -> m [i] [j] : (1, 3, 12)  
m0 -> t [i]      : (1, 31, 0)  
m1 -> m [i] [j] : (1, 3, 12)  
m1 -> t [i]      : (1, 31, 0)  
m2 -> m [i] [j] : (1, 3, 12)  
m2 -> t [i]      : (1, 31, 0)
```

This function destroys a rotation matrix.

Return value

```
m2
```

See also

```
CompMatrix()
```


csin

C sine function.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
int csin(
int a)                Angle (in PlayStation format)
```

Explanation

Find the sine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2\pi$) using fixed point math (where $4096 = 1.0$).

Compared to `rsin()`, `csin()` is slower and takes up less space.

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$)

Return value

`sin(a)` : (1, 19, 12)

See also

[rsin\(\)](#), [ccos\(\)](#), [catan\(\)](#)

csqrt

C square root function.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
int csqrt(
int a)           Value
```

Explanation

Uses fixed point math (where 4096 = 1.0) to find the fixed point square root.

This function is the same as SquareRoot12() except that it requires a smaller table memory area.

a : (1, 19, 12)

Return value

sqrt (*a*) : (1, 19, 12)

See also

[SquareRoot0\(\)](#), [SquareRoot12\(\)](#), [InvSquareRoot\(\)](#)

DivideF3, DivideF4, DivideFT3, DivideFT4, DivideG3, DivideG4, DivideGT3, DivideGT4

Polygon division functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

Flat triangle.

```
u_long *DivideF3(SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, CVECTOR *rgbc, POLY_F3 *s, u_long *ot, DIVPOLYGON3 *divp)
```

Flat quadrilateral.

```
u_long *DivideF4( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, SVECTOR *v3, CVECTOR *rgbc, POLY_F4 *s, u_long *ot, DIVPOLYGON4 *divp)
```

Flat textured triangle.

```
u_long *DivideFT3( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, u_long *uv0, u_long *uv1, u_long *uv2, CVECTOR *rgbc, POLY_FT3 *s, u_long *ot, DIVPOLYGON3 *divp)
```

Flat textured quadrilateral.

```
u_long *DivideFT4( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, SVECTOR *v3, u_long *uv0, u_long *uv1, u_long *uv2, u_long *uv3, CVECTOR *rgbc, POLY_FT4 *s, u_long *ot, DIVPOLYGON4 *divp)
```

Gouraud-shaded triangle.

```
u_long *DivideG3( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, CVECTOR *rgb0, CVECTOR *rgb1, CVECTOR *rgb2, POLY_G3 *s, u_long *ot, DIVPOLYGON3 *divp)
```

Gouraud-shaded quadrilateral.

```
u_long *DivideG4( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, SVECTOR *v3, CVECTOR *rgb0, CVECTOR *rgb1, CVECTOR *rgb2, CVECTOR *rgb3, POLY_G4 *s, u_long *ot, DIVPOLYGON4 *divp)
```

Gouraud-shaded, textured triangle.

```
u_long *DivideGT3( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, u_long *uv0, u_long *uv1, u_long *uv2, CVECTOR *rgb0, CVECTOR *rgb1, CVECTOR *rgb2, POLY_GT3 *s, u_long *ot, DIVPOLYGON3 *divp)
```

Gouraud-shaded textured quadrilateral.

```
u_long *DivideGT4( SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, SVECTOR *v3, u_long *uv0, u_long *uv1, u_long *uv2, u_long *uv3, CVECTOR *rgb0, CVECTOR *rgb1, CVECTOR *rgb2, CVECTOR *rgb3, POLY_GT4 *s, u_long *ot, DIVPOLYGON4 *divp)
```

Explanation

Divides a polygon and registers the result to the OT.

v0 – *v3* are pointers to the vertex coordinate vectors.

uv0 – *uv3*, for textured polygons, are pointers to the texture coordinate vectors.

rgbc is a pointer to a color vector + code.

s is a pointer to the GPU packet buffer address. It depends on the polygon type.

ot is a pointer to the OT entry

divp is a pointer to the division work area. You must set *divp->ndiv* to the desired number of divisions, and *divp->pih*, *divp->piv* to the display screen (clipping) resolution.

The *divp* -> *ndiv* values and division format are shown below:

Table 8-1: Division types

<i>divp->ndiv</i>	Number of divisions
1	2x2
2	4x4
3	8x8
4	16 x 16
5	32 x 32

rgb0 – *rgb3*, for Gouraud-sh

aded polygons, are pointers to color vectors.

rgb0:rgb0+code

Return value

Updated GPU packet buffer address.

DpqColor

Interpolate a primary color vector and far color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void DpqColor(  
  CVECTOR *v0,           Pointer to primary color vector (input)  
  long p,                 Interpolation value (input)  
  CVECTOR *v1)            Pointer to primary color vector (output)
```

Explanation

Calculates $v1 = (1-p) \times v0 + p \times FC$.

v0 -> r, g, b : (0, 8, 0)
p : (0, 20, 12)
v1 -> r, g, b : (0, 8, 0)

See also

[DpqColor3\(\)](#), [DpqColorLight\(\)](#)

DpqColor3

Interpolate three primary color vectors and far color.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void DpqColor3(
CVECTOR *v0, CVECTOR *v1, CVECTOR *v2, Pointer to primary color vectors (input)
long p, Interpolation value (input)
CVECTOR *v3, CVECTOR *v4, CVECTOR *v5) Pointer to color vectors (output)

Explanation

Calculates:

$v3 = (1-p) \times v0 + p \times FC$
 $v4 = (1-p) \times v1 + p \times FC$
 $v5 = (1-p) \times v2 + p \times FC.$
 $v0, v1, v2 \rightarrow r, g, b : (0, 8, 0)$
 $p : (0, 20, 12)$
 $v3, v4, v5 \rightarrow r, g, b : (0, 8, 0)$

See also

DpqColor(), DpqColorLight()

DpqColorLight

Interpolate the product from multiplication of a local color vector by primary color vector, and far color.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

```
void DpqColorLight(  
SVECTOR *v0,           Pointer to local color vector (input)  
CVECTOR *v1,           Pointer to primary color vector (input)  
long p,                Interpolation value (input)  
CVECTOR *v2)           Pointer to color vector (output)
```

Explanation

Calculates $v2 = (1-p) \times (v1 \times v0) + p \times FC$.

where $v1 \times v0$ is a separate multiplication product.

```
v0 -> vx, vy, vz      : (1, 3, 12)  
v1 -> r, g, b         : (0, 8, 0)  
p                     : (0, 20, 12)  
v2 -> r, g, b         : (0, 8, 0)
```

See also

DpqColor(), DpqColor3()

EigenMatrix

Obtain the eigen matrix.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.1	12/14/98

Syntax

```
void EigenMatrix(  
  MATRIX *m,           Input: rotation matrix  
  MATRIX *t)           Output: eigen matrix
```

Explanation

The eigen matrix (ordered eigen vectors) corresponding to the input rotation matrix, *m*, is output to the matrix *t*.

The operation performed is shown below.

$$[t]^{-1} * [m] * [t]^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

```
m -> m [1] [1] : (1, 3, 12)  
t -> m [1] [1] : (1, 3, 12)
```


gteMIMefunc

Add a vertex data array to a differential data array multiplied by a coefficient.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void gteMIMefunc(
SVECTOR *otp, Pointer to a vertex array
SVECTOR *dfp, Pointer to a differential array
long n, Number of vertex (differential) data
long p) Weight (control) coefficient: (1, 19, 12)

Explanation

Executes calculation of multiple interpolations using vertex data array and difference data array. The argument format is as follows:

p : (1, 19, 12)

otp, *dfp* optional

It operates at high speed in a similar way to the program given in the example below.

```
void gteMIMefunc (otp, dfp, n, p)
SVECTOR *otp, *dfp;
long n, p;
{
  int i;
  for (i = 0; i<n; i++) {
    (otp+i)->x+=((int)((dfp+i)->x) x p)>>12;
    (otp+i)->y+=((int)((dfp+i)->y) x p)>>12;
    (otp+i)->z+=((int)((dfp+i)->z) x p)>>12;
  }
}
```

InitClip

Initialize clipping parameter.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
void InitClip(
  EVECTOR *evbfad,    Pointer to 16 clip vector data arrays
  long hw,             Window width
  long vw,             Window height
  long h,              Projection distance from view point to screen
  long near,           NearClip position
  long far)            FarClip position
```

Explanation

Sets parameters used for clipping.

The clip vector data array *evbfad* reserves 16 data arrays (176 words or 704 bytes).

See also

[Clip3F\(\)](#), [Clip4F\(\)](#)

InitGeom

Initialize the geometry transform engine.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void InitGeom(*void*)

Explanation

Initializes the GTE. It must be called whenever the basic geometry library is used.

Intpl

Interpolate a vector and far color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void Intpl(
  SVECTOR *v0,      Pointer to vector (input)
  long p,           Interpolation value (input)
  CVECTOR *v1)      Pointer to vector (output)
```

Explanation

Calculates $v1 = (1-p) \times v0 + p \times FC$.

```
v0 -> vx, vy, vz      : (1, 3, 12)
p                      : (0, 20, 12)
v1 -> r, g, b          : (0, 8, 0)
```

InvSquareRoot

Inverse square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void InvSquareRoot(

long *a*, Value
long **b*, Pointer to mantissa
long **c*) Pointer to exponent

Explanation

Calculates 1/square root of *a*.

a : (0, 32, 0)
b : (0, 20, 12)
c : (0, 32, 0)

If *a* > 0x7FFFFFFF, a processor exception will occur.

See also

[csqrt\(\)](#), [SquareRoot12\(\)](#)

IsIdMatrix

Judge distance from unit matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void IsIdMatrix(  
MATRIX *m)           Input matrix
```

Explanation

Compares the input matrix *m* with the unit matrix. If the elements of matrix *m* are less than 20 away from the unit matrix, the function returns the value 1.

m -> *m* [i] [j] : (1, 3, 12)

Return value

1 if the matrix is the unit matrix, else 0.

LightColor

Coordinate transformation using the local color matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void LightColor(

SVECTOR *v0,

Pointer to vector (input)

VECTOR *v1)

Pointer to vector (output)

Explanation

Calculates $v1 = LCM * v0$. A limiter works on negative components of $v1$ when 0 is reached.

$v0 \rightarrow vx, vy, vz$: (1, 3, 12)

$v1 \rightarrow vx, vy, vz$: (0, 20, 12)

LoadAverage0

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
void LoadAverage0(
  VECTOR *v0, VECTOR *v1,    Pointer to vectors (input)
  long p0, long p1,          Weights (input)
  VECTOR *v2)                Pointer to vector (output)
```

Explanation

Returns the weighted average of two vectors *v0* and *v1* in *v2* using weights of *p0* and *p1*.

```
v0, v1 -> vx, vy, vz      : (1, 31, 0)
p0, p1                     : (1, 15, 0)
v2 -> vx, vy, vz          : (1, 31, 0)
```

See also

[LoadAverage12\(\)](#), [LoadAverageByte\(\)](#), [LoadAverageCol\(\)](#), [LoadAverageShort0\(\)](#), [LoadAverageShort12\(\)](#)

LoadAverage12

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void LoadAverage12(
VECTOR *v0, VECTOR *v1,    Pointer to vectors (input)
long p0, long p1,          Weights (input)
VECTOR *v2)                Pointer to vector (output)
```

Explanation

Finds the weighted average of two vectors *v0* and *v1* using weights of *p0* and *p1* after division by 4096 (1 in fixed point format) the results are returned in *v2*.

```
v0, v1 -> vx, vy, vz      : (1, 31, 0)
p0, p1                    : (1, 3, 12)
v2 -> vx, vy, vz          : (1, 31, 0)
```

See also

[LoadAverage0\(\)](#), [LoadAverageByte\(\)](#), [LoadAverageCol\(\)](#), [LoadAverageShort0\(\)](#), [LoadAverageShort12\(\)](#)

LoadAverageByte

Find weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
void LoadAverageByte(  
u_char v0[2], u_char v1[2],      Vector (input)  
long p0, p1,                    Weights (input)  
u_char v2[2])                   Vector (output)
```

Explanation

Finds the weighted average of two vectors *v0* and *v1* using weights *p0* and *p1*. The result is returned in *v2* after division by 4096.

```
v0[i], v1[i]      : (0, 8, 0)  
p0, p1            : (1, 3, 12)  
v2[i]             : (0, 8, 0)
```

See also

[LoadAverage0\(\)](#), [LoadAverage12\(\)](#), [LoadAverageCol\(\)](#), [LoadAverageShort0\(\)](#), [LoadAverageShort12\(\)](#)

LoadAverageCol

Find weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
void LoadAverageCol(  
u_char v0[3], u_char v1[3],      Vectors (input)  
long p0, long p1,                Weights (input)  
u_char v2[3])                   Vector (output)
```

Explanation

Finds the weighted average of two vectors *v0* and *v1* using weights *p0* and *p1*. The result is returned in *v2* after division by 4096.

```
v0[i], v1[i]      : (0, 8, 0)  
p0, p1            : (1, 3, 12)  
v2[i]             : (0, 8, 0)
```

See also

[LoadAverage0\(\)](#), [LoadAverage12\(\)](#), [LoadAverageByte\(\)](#), [LoadAverageShort0\(\)](#), [LoadAverageShort12\(\)](#)

LoadAverageShort0

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void LoadAverageShort0(
SVECTOR *v0, SVECTOR *v1, Pointer to vectors (input)
long p0, long p1, Weights (input)
SVECTOR *v2) Pointer to vector (output)

Explanation

Returns the weighted average of two vectors v0 and v1 in v2 using weights of p0 and p1.

v0, v1 -> vx, vy, vz : (1, 15, 0)
p0, p1 : (1, 15, 0)
v2 -> vx, vy, vz : (1, 30, 0)

See also

[LoadAverage0\(\)](#), [LoadAverage120\(\)](#), [LoadAverageByte\(\)](#), [LoadAverageCol\(\)](#), [LoadAverageShort120\(\)](#)

LoadAverageShort12

Weighted average of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void LoadAverageShort12(
  SVECTOR *v0, SVECTOR *v1,    Pointer to vectors (input)
  long p0, long p1,           Weights (input)
  SVECTOR *v2)                Pointer to vector (output)
```

Explanation

Finds the weighted average of two vectors *v0* and *v1* using weights of *p0* and *p1* after division by 4096 (1 in fixed point format) the results are returned to *v2*.

```
v0, v1 -> vx, vy, vz          : (1, 15, 0)
p0, p1                          : (1, 3, 12)
v2 -> vx, vy, vz               : (1, 15, 0)
```

See also

[LoadAverage0\(\)](#), [LoadAverage12\(\)](#), [LoadAverageByte\(\)](#), [LoadAverageCol\(\)](#), [LoadAverageShort0\(\)](#)

LocalLight

Coordinate transformation using the local light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void LocalLight(

SVECTOR *v0,

Pointer to vector (input)

VECTOR *v1)

Pointer to vector (output)

Explanation

Calculates $v1 = LLM * v0$. A limiter works on negative components of $v1$ when 0 is reached.

$v0 \rightarrow vx, vy, vz:$:(1, 3, 12)

$v1 \rightarrow vx, vy, vz:$:(0, 20, 12)

Lzc

Calculate leading zero count.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long Lzc(  
long data)           Value
```

Explanation

Calculates the Leading Zero Count (LZC) given by *data*.

data : (1, 31, 0)

Return value

The value of LZC.

MatrixNormal

Normalize a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

```
void MatrixNormal(  
  MATRIX *m,           Pointer to matrix (input)  
  MATRIX *n)           Pointer to matrix (output)
```

Explanation

Orthonormalizes a distorted rotation matrix.

(Do not use m[2][0], m[2][1], m[2][2].)

m -> m [i] [j] : (1, 3, 12)
n -> m [i] [j] : (1, 3, 12)

See also

[MatrixNormal_0\(\)](#), [MatrixNormal_1\(\)](#), [MatrixNormal_2\(\)](#)

MatrixNormal_0

Orthonormalize a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.6	12/14/98

Syntax

void MatrixNormal_0(

*MATRIX *m*, Pointer to matrix (input)
*MATRIX *n*) Pointer to matrix (output)

Explanation

Orthonormalizes a distorted rotation matrix.

(Do not use m[2][0], m[2][1], m[2][2].)

m -> m [i] [j] : (1, 3, 12)
n -> m [i] [j] : (1, 3, 12)

See also

[MatrixNormal_1\(\)](#), [MatrixNormal_2\(\)](#)

MatrixNormal_1

Normalize a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

void MatrixNormal_1(

MATRIX *m, Pointer to matrix (input)
MATRIX *n) Pointer to matrix (output)

Explanation

Orthonormalizes a distorted rotation matrix.

(Do not use m[0][0], m[0][1], m[0][2].)

m -> m [i] [j] (1, 3, 12)
n -> m [i] [j] (1, 3, 12)

See also

[MatrixNormal_0\(\)](#), [MatrixNormal_2\(\)](#)

MatrixNormal_2

Normalize a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

void MatrixNormal_2(

MATRIX *m, Pointer to matrix (input)
MATRIX *n) Pointer to matrix (output)

Explanation

Orthonormalizes a distorted rotation matrix.

(Do not use m[1][0], m[1][1], m[1][2].)

m -> m [i] [j] : (1, 3, 12)
n -> m [i] [j] : (1, 3, 12)

See also

[MatrixNormal_0\(\)](#), [MatrixNormal_1\(\)](#)

MulMatrix

Multiply two matrices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

MATRIX *MulMatrix(
MATRIX *m0, MATRIX *m1) Pointer to input/output matrices

Explanation

Multiplies two matrices. The result is saved in *m0*.

m0, m1 -> m [i] [j] : (1, 3, 12)

The function destroys the constant rotation matrix.

Return value

m0.

See also

[MulMatrix0\(\)](#), [MulMatrix2\(\)](#), [MulRotMatrix\(\)](#), [MulRotMatrix0\(\)](#)

MulMatrix0

Multiply two matrices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
MATRIX *MulMatrix0(
MATRIX *m0, MATRIX *m1,    Pointer to input matrices
MATRIX *m2)                Pointer to output matrix
```

Explanation

Multiplies two matrices *m0* and *m1*.

m0, *m1*, *m2* -> m [i] [j] : (1, 3, 12)

The function destroys the constant rotation matrix.

Return value

m2.

See also

[MulMatrix\(\)](#), [MulMatrix2\(\)](#), [MulRotMatrix\(\)](#), [MulRotMatrix0\(\)](#)

MulMatrix2

Multiply two matrices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

MATRIX *MulMatrix2(
MATRIX *m0, MATRIX *m1) Pointer to input/output matrices

Explanation

Multiplies two matrices. The result is saved in *m1*.

m0, m1 -> m [i] [j] : (1, 3, 12)

The function destroys the constant rotation matrix.

Return value

m1.

See also

[MulMatrix\(\)](#), [MulMatrix0\(\)](#), [MulRotMatrix\(\)](#), [MulRotMatrix0\(\)](#)

MulRotMatrix

Multiply a constant rotation matrix by a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
MATRIX *MulRotMatrix(  
MATRIX *m0)           Pointer to input/output matrix
```

Explanation

Multiplies a constant rotation matrix by a matrix. It stores the value in *m0*.

m0, m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m0*.

See also

[MulMatrix\(\)](#), [MulMatrix0\(\)](#), [MulMatrix2\(\)](#), [MulRotMatrix0\(\)](#)

MulRotMatrix0

Multiply a constant rotation matrix by a matrix.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	3.0	12/14/98

Syntax

```
MATRIX *MulRotMatrix0(  
  MATRIX *m0,           Pointer to input matrix  
  MATRIX *m1)           Pointer to output matrix
```

Explanation

Multiplies a constant rotation matrix by matrix *m0*. The result is saved in *m1*.

m0, m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m1*.

See also

MulMatrix(), MulMatrix0(), MulMatrix2(), MulRotMatrix()

NormalClip

Outer product of three points.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long NormalClip(

long *sxy0*, **long** *sxy1*, **long** *sxy2*) Input values (upper 16 bits are screen x coordinates (sx); lower 16 bits are screen y coordinates (sy))

Explanation

Returns the outer product for a triangle formed by three points (*sx0*, *sy0*), (*sx1*, *sy1*), and (*sx2*, *sy2*).

When viewed from the direction of the viewpoint (Z axis negative) the value is positive when the triangle is righthanded.

(However, the X axis positive faces right and the Y axis positive faces down.)

sxy0, *sxy1*, *sxy2* : (1, 15, 0), (1, 15, 0)

Return value

| *sx1-sx0*, *sy1-sy0* |

| *sx2-sx0*, *sy2-sy0* |

See also

[OuterProduct0\(\)](#), [OuterProduct12\(\)](#)

NormalColor, NormalColor_nom

Find a local color from a normal vector.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void NormalColor(
SVECTOR *v0, Pointer to normal vector (input)
CVECTOR *v1) Pointer to color vector (output)

void NormalColor_nom(
SVECTOR *v0) Pointer to normal vector (input)

Explanation

LLV = LLM * v0

v1 = BK + LCM * LLV

v0 -> vx, vy, vz : (1, 3, 12)

NormalColor(): v1-> r, g, b : (0, 8, 0)

NormalColor_nom():The operation result(s) must be retrieved from the GTE. (For further information see the Inline Reference documentation.)

- For (v1->r,v1->g, v1->b) use the read_rgb2 macro.

See also

NormalColor3(), NormalColorCol(), NormalColorCol3(), NormalColorDpq(), NormalColorDpq3()

NormalColor3, NormalColor3_nom

Find three local colors from three normal vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void NormalColor3(

SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, Pointer to normal vectors (input)
CVECTOR *v3, **CVECTOR** *v4, **CVECTOR** *v5) Pointer to color vectors (output)

```
void NormalColor3_nom(
```

SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2) Pointer to normal vectors (input)

Explanation

$$(LLV0, LLV1, LLV2) = LLM * (v0, v1, v2)$$
$$(v3, v4, v5) = BK + LCM * (LLV0, LLV1, LLV2)$$
$$v_0, v_1, v_2 \rightarrow v_x, v_y, v_z \quad : (1, 3, 12)$$

NormalColor3(): v3, v4, v5 -> r, g, b : (0, 8, 0)

NormalColor3_nom():The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- For (v3,v4 and v5) use the read_rgb_fifo macro.

See also

NormalColor(), NormalColorCol(), NormalColorCol3(), NormalColorDpq(), NormalColorDpq3()

NormalColorCol, NormalColorCol_nom

Find a local color from a normal vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void NormalColorCol(
SVECTOR *v0, Pointer to normal vector (input)
CVECTOR *v1, Pointer to primary color vector (input)
CVECTOR *v2) Pointer to color vector (output)

void NormalColorCol_nom(
SVECTOR *v0, Pointer to normal vector (input)
CVECTOR *v1) Pointer to primary color vector (input)

Explanation

$LLV = LLM * v0$

$LC = BK + LCM * LLV$

$v2 = v1 * LC$

v0 -> vx, vy, vz : (1, 3, 12)

v1 -> r, g, b : (0, 8, 0)

NormalColorCol(): v2 -> r, g, b : (0, 8, 0)

NormalColorCol_nom():The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- For (v2->r,v2->g,v2->b) use the read_rgb2 macro.

See also

NormalColor(), NormalColor3(), NormalColorCol3(), NormalColorDpq(), NormalColorDpq3()

NormalColorCol3, NormalColorCol3_nom

Find a local color from three normal vectors.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void NormalColorCol3(
SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,
CVECTOR *v3,
CVECTOR *v4, CVECTOR *v5, CVECTOR *v6)

Pointer to normal vectors (input)
Pointer to primary color vector (input)
Pointer to color vectors (output)

void NormalColorCol3_nom(
SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,
CVECTOR *v3)

Pointer to normal vectors (input)
Pointer to primary color vector (input)

Explanation

(LLV0, LLV1, VVL2) = LLM * (v0, v1, v2)
(LC0, LC1, LC2) = BK + LCM * (LLV0, LLV1, LLV2)
(v4, v5, v6) = v3 * (LC0, LC1, LC2)
v0, v1, v2 -> vx, vy, vz : (1, 3, 12)
v3 -> r, g, b : (0, 8, 0)
NormalColorCol3(): v4, v5, v6 -> r, g, b : (0, 8, 0)

NormalColorCol3_nom():The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- For (v4,v5,v6) use the read_rgb_fifo macro.

See also

[NormalColor\(\)](#), [NormalColor3\(\)](#), [NormalColorCol\(\)](#), [NormalColorDpq\(\)](#), [NormalColorDpq3\(\)](#)

NormalColorDpq, NormalColorDpq_nom

Find a local color from a normal vector and perform depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void NormalColorDpq(
 SVECTOR *v0, Pointer to normal vector (input)
 CVECTOR *v1, Pointer to primary color vector (input)
 long p, Interpolation value (input)
 CVECTOR *v2) Pointer to color vector (output)

void NormalColorDpq_nom(
 SVECTOR *v0, Pointer to normal vector (input)
 CVECTOR *v1, Pointer to primary color vector (input)
 long p) Interpolation value (input)

Explanation

$LLV = LLM * v0$
 $LC = BK + LCM * LLV$
 $v2 = (1 - p) * v1 * LC + p * FC$
 $v0 \rightarrow vx, vy, vz$: (1, 3, 12)
 $v1 \rightarrow r, g, b$: (0, 8, 0)
 p : (0, 20, 12)
NormalColorDpq(): $v2 \rightarrow r, g, b$: (0, 8, 0)

NormalColorDpq_nom():The operation result(s) must be retrieved from the GTE. (For further information see the Inline Reference documentation.):

- For (v2->r,v2->g,v2->b) use the read_rgb2 macro.

See also

[NormalColor\(\)](#), [NormalColor3\(\)](#), [NormalColorCol\(\)](#), [NormalColorCol3\(\)](#), [NormalColorDpq3\(\)](#)

NormalColorDpq3, NormalColorDpq3_nom

Find local color from three normal vectors and perform depth cueing.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void NormalColorDpq3(
SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2,
CVECTOR *v3,
long p,
CVECTOR *v4, **CVECTOR** *v5, **CVECTOR** *v6)

Pointer to normal vectors (input)

Pointer to primary color vector (input)

Interpolation value (input)

Pointer to color vectors (output)

void NormalColorDpq3_nom(
SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2,
CVECTOR *v3,
long p)

Pointer to normal vectors (input)

Pointer to primary color vector (input)

Interpolation value (input)

Explanation

$(LLV0, LLV1, LLV2) = LLM * (v0, v1, v2)$
 $(LC0, LC1, LC2) = BK + LCM * (LLV0, LLV1, LLV2)$
 $(v4, v5, v6) = (1 - p) * v3 * (LC0, LC1, LC2) + p * FC$
 $v0, v1, v2 \rightarrow vx, vy, vz$: (1, 3, 12)
 $v3 \rightarrow r, g, b$: (0, 8, 0)
 p : (0, 20, 12)
 $NormalColorDpq3(): v4, v5, v6 \rightarrow r, g, b$: (0, 8, 0)

NormalColorDpq3_nom():The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- For (v4,v5,v6) use the read_rgb_fifo macro.

See also

NormalColor(), NormalColor3(), NormalColorCol(), NormalColorCol3(), NormalColorDpq()

otz2p

Get depth cueing interpolation value p from OTZ value.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long otz2p(
long otz,           OTZ
long projection)    Distance between visual point and screen
```

Explanation

Get the approximate depth cueing interpolation value p from otz , which is $1/4$ of sz , the z element of the screen coordinates.

Depending on the fog setting, errors can increase and the results are not necessarily the same as with RotTransPers() functions.

Return value

Depth cueing interpolation value p (0: 0%, 4096 : 100%).

See also

[p2otz\(\)](#)

OuterProduct0

Outer product of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void OuterProduct0(
  VECTOR *v0, VECTOR *v1,    Pointer to vectors (input)
  VECTOR *v2)                Pointer to vector (output)
```

Explanation

Returns the outer product vector of two vectors *v0* and *v1* to *v2*.

v0, *v1* -> vx, vy, vz : (1, 31, 0)
v2 -> vx, vy, vz : (1, 31, 0)

See also

[NormalClip\(\)](#), [OuterProduct12\(\)](#)

OuterProduct12

Outer product of two vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void OuterProduct12(
[VECTOR](#) *v0, **VECTOR** *v1, Pointer to vectors (input)
[VECTOR](#) *v2) Pointer to vector (output)

Explanation

Returns the outer product vector of two vectors, v0 and v1, to v2.

v0, v1, v2 -> vx, vy, vz : (1, 19, 12)

See also

[NormalClip\(\)](#), [OuterProduct0\(\)](#)

p2otz

Get otz from depth cueing interpolation value.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long p2otz(

long *p*, Interpolation value (0 to 4096)

long *projection*) Distance between visual point and screen

Explanation

Gets the *otz* value, which is 1/4 of *sz* (screen coordinate *z* element) from the depth cueing interpolation value *p*.

Depending on the fog setting, errors can increase and the results are not necessarily the same as with RotTransPers() functions.

otz when *P*=0 or *p*=4096 is not theoretically decided as identification, but with this function a convenient value is returned.

Return value

OTZ value.

See also

[otz2p\(\)](#)

pers_map

Perspective conversion texture mapping.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

```
void pers_map(
int abuf,           ID of displayed buffer
SVECTOR **vertex,  3 dimensional coordinates of 4 vertices
int tex[4][2],     Texture address of 4 vertices
u_short *dtext)    Pointer to texture storage location converted to direct color
```

Explanation

Performs texture mapping with no distortion.

Flat texture, with no light source calculations only.

The 4 vertices are only square, rectangle and parallelogram locations.

Z sort by OT is not possible.

PhongLine

Phong shading.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

void PhongLine(

int <i>istart_x</i> ,	X coordinate of starting point
int <i>iend_x</i> ,	X coordinate of finishing point
int <i>p</i> ,	Differential X coordinate of fs value
int <i>q</i> ,	Differential caused by X coordinate of ft value
u_short * <i>pixx</i> ,	Pixel pointer
int <i>fs</i> ,	Interpolation coefficient at start point
int <i>ft</i> ,	Interpolation coefficient at start point
int <i>i4</i> ,	(Line number) %4 due to dithering
int <i>def</i>)	Queue method of edge queue

Explanation

Performs one line Phong shading. For more information, refer to sample program (sample/graphics/phong)

PopMatrix

Reset a constant rotation matrix from a stack.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void PopMatrix(*void*)

Explanation

Resets a constant rotation matrix from a stack.

See also

[PushMatrix\(\)](#)

PushMatrix

Save a constant rotation matrix in a stack.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void PushMatrix(*void*)

Explanation

Saves a constant rotation matrix on a stack. The stack has 20 slots.

See also

[PopMatrix\(\)](#)

ratan2

Arctan.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
long ratan2(
long x, long y)      Value
```

Explanation

Uses PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$) to finish the x/y arctan function (-180 degrees and $+180 \text{ degrees}$, $-\pi \dots \pi$).

Return value

atan2 (x, y) : (1, 19, 12)

The return value is incorrect if either x or y is -2147483648 ($0x80000000 = \text{long negative maximum value}$).

See also

[catan\(\)](#), [rcos\(\)](#), [rsin\(\)](#)

rcos

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
int rcos(
int a)                Angle (in PlayStation format)
```

Explanation

Finds the cosine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2\pi$) using fixed-point math (where $4096=1.0$).

Compared to `ccos()`, `rcos()` is faster and takes up more space.

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$)

Return value

`cos (a)` : (1, 19, 12)

See also

[ccos\(\)](#), [ratan2\(\)](#), [rsin\(\)](#)

RCpolyF3, RCpolyFT3, RCpolyG3, RCpolyGT3

Triangle division functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

<code>u_long *RCpolyF3(POLY_F3 *s, DIVPOLYGON3 *divp)</code>	Flat triangle.
<code>u_long *RCpolyFT3(POLY_FT3 *s, DIVPOLYGON3 *divp)</code>	Flat, textured triangle.
<code>u_long *RCpolyG3(POLY_G3 *s, DIVPOLYGON3 *divp)</code>	Gouraud-shaded triangle
<code>u_long *RCpolyGT3(POLY_GT3 *s, DIVPOLYGON3 *divp)</code>	Gouraud-shaded, textured triangle

Explanation

These are recursive functions for division of triangles. *s* is a pointer to the GPU packet buffer address. *divp* is a pointer to a division work area. You must set the data below in the *divp* work area:

<code>u_long ndiv</code>	Number of divisions
<code>u_long pih, piv</code>	Display screen resolution (for clipping)
<code>u_short clut, tpage</code>	CBA & TSB
<code>CVECTOR rgbc</code>	Color vector (+code)
<code>u_long *ot</code>	OT entry
<code>RVECTOR r0, r1, r2</code>	Division vertex vector data
<code>CRVECTOR3 cr[5]</code>	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of `CRVECTOR3 (cr[5])` to the value of the vertex vector data of `RVECTORs r0`, `r1`, and `r2`.

Note: See [DIVPOLYGON3](#) for a full description of *divp*.

Return value

Updated GPU packet buffer address

See also

[RCpolyF4\(\)](#)

RCpolyF4, RCpolyFT4, RCpolyG4, RCpolyGT4

Quadrilateral division functions.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	3.0	12/14/98

Syntax

u_long *RCpolyF4(POLY_F4 *s,DIVPOLYGON4 *divp)	Flat quadrilateral.
u_long *RCpolyFT4(POLY_FT4 *s,DIVPOLYGON4 *divp)	Flat, textured quadrilateral.
u_long *RCpolyG4(POLY_G4 *s,DIVPOLYGON4 *divp)	Gouraud-shaded quadrilateral
u_long *RCpolyGT4(POLY_GT4 *s,DIVPOLYGON4 *divp)	Gouraud-shaded, textured quadrilateral

Explanation

These are recursive functions for division of quadrilaterals. s is a pointer to the GPU packet buffer address. divp is a pointer to a division work area. You must set the data below in the divp work area:

u_long ndiv	Number of divisions
u_long pih, piv	Display screen resolution (for clipping)
u_short clut, tpage	CBA & TSB
CVECTOR rgbc	Color vector (+code)
u_long *ot	OT entry
RVECTOR r0, r1, r2	Division vertex vector data
CRVECTOR4 cr[5]	2D and 3D texture coordinates and color for each vertex

Assign the vertex vector data of CRVECTOR4 (cr[5]) to the value of the vertex vector data of RVECTORs r0, r1, r2 and r3.

Note: See DIVPOLYGON4 for a full description of divp.

Return value

Updated GPU packet buffer address

See also

RCpolyF3()

ReadColorMatrix

Read a local color matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void ReadColorMatrix(
    MATRIX *m)           Pointer to matrix (input)
```

Explanation

Reads the current local color matrix, and saves it in *m*.

m -> m [i] [j] : (1, 3, 12)

ReadGeomOffset

Read GTE offset value.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

void ReadGeomOffset(

long *ofx, Pointer to offset X coordinate

long *ofy) Pointer to offset Y coordinate

Explanation

Reads the GTE offset value.

ofx, ofy : (0, 32, 0)

ReadGeomScreen

Read distance from viewpoint to screen.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

long

ReadGeomScreen(*void*)

Explanation

Reads the distance h from the viewpoint (eye) to the screen.

Return value

h value

ReadLightMatrix

Read a local light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void ReadLightMatrix(
MATRIX **m*) Pointer to matrix (input)

Explanation

Reads the current local light matrix, and saves it in *m*.

m -> m [i] [j] : (1, 3, 12)

ReadRGBfifo

Read RGBcd values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void ReadRGBfifo(
[CVECTOR](#) *v0, [CVECTOR](#) *v1, [CVECTOR](#) *v2)** Pointer to vectors (output)

Explanation

Stores the RGBcd0, RGBcd1, and RGBcd2 values in v0, v1, and v2.

v0, v1, v2 -> r, g, b, cd: (0, 8, 0)

See also

[ReadSXSIfifo\(\)](#), [ReadSZfifo3\(\)](#), [ReadSZfifo4\(\)](#)

ReadRotMatrix

Read a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void ReadRotMatrix(
 MATRIX *m)** Pointer to matrix (output)

Explanation

Reads the current rotation matrix, and saves it in *m*.

m -> m [i] [j] :(1, 3, 12)
m -> t [i] : (1, 31, 0)

ReadSXSyfifo

Read SXSy values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void ReadSXSyfifo(
long *sxy0, long*sxy1, long *sxy2)** Pointers to screen x,y values

Explanation

Stores the sx0, sy0, sx1, sy1, sx2, and sy2 values in sxy0, sxy1, and sxy2.

(sxy0, sxy1, sxy2) : (1, 15, 0)

See also

[ReadRGBfifo\(\)](#), [ReadSZfifo3\(\)](#), [ReadSZfifo4\(\)](#)

ReadSZfifo3

Read SZ values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void ReadSZfifo3(
long *sz0, *sz1, *sz2)** Pointers to SZ values

Explanation

Stores the sz0, sz1, and sz2 values in sz0, sz1, and sz2.

(sz0, sz1, sz2) : (0, 16, 0)

See also

[ReadRGBfifo\(\)](#), [ReadSXSyfifo\(\)](#), [ReadSZfifo4\(\)](#)

ReadSZfifo4

Read SZ values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void ReadSZfifo4(
long *szx, long *sz0, long *sz1, long *sz2)** Pointers to SZ values

Explanation

Stores the szx, sz0, sz1, and sz2 values in szx, sz0, sz1, and sz2.
(szx, sz0, sz1, sz2) : (0, 16, 0)

See also

[ReadRGBfifo\(\)](#), [ReadSXSyfifo\(\)](#), [ReadSZfifo3\(\)](#)

RotAverage3, RotAverage3_nom

Perform coordinate and perspective transformation for 3 points, and get interpolation value and average of Z values for depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long RotAverage3(
  SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,    Pointers to vectors (input)
  long *sxy0, long *sxy1, long *sxy2,        Pointers to coordinates (output)
  long *p,                                    Pointers to interpolation values (output)
  long *flag)                                Pointer to flag (output)
```

```
long RotAverage3_nom
(SVECTOR *v0, SVECTOR *v1, SVECTOR *v2)    Pointer to vectors (input)
```

Explanation

A coordinate transformation of three points *v0*, *v1*, *v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sxy0*, *sxy1*, and *sxy2* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. The return value becomes the average of three screen coordinate Z values.

```
v0, v1, v2 -> vx, vy, vz    : (1, 15, 0)
sxy0, sxy1, sxy2            : (1, 15, 0), (1, 15, 0)
p                            : (0, 20, 12)
flag                        : (0, 32, 0)
```

RotAverage3_nom():The operation result(s) must be retrieved from the GTE. (For further information, refer to the Inline Reference documentation.)

- For (sz0,sz1,sz2) use macro read_sz_fifo3
- For ((sx0,sy0),(sx1,sy1),(sx2,sy2)) use macro read_sxsy_fifo3
- For p use macro read_p
- For otz use macro read_otz.
- flag is returned in register v0.

Return value

OTZ value

(RotAverage3_nom) returns no value)

See also

[RotAverage4\(\)](#), [RotAverageNclip3\(\)](#), [RotAverageNclipColorCol3\(\)](#), [RotAverageNclipColorDpq3\(\)](#), [RotColorDpq3\(\)](#)

RotAverage4

Perform coordinate transformation for 3 points and perspective transformation, and find an interpolation value and an average of Z values for depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotAverage4(

SVECTOR *v0, SVECTOR *v1, SVECTOR *v2, SVECTOR *v3,	Pointer to vectors (input)
long *sxy0, long *sxy1, long *sxy2, long *sxy3,	Pointer to coordinates (output)
long *p,	Pointer to interpolation (output)
long *flag)	Pointer to flag (output)

Explanation

A coordinate transformation of four points v0, v1, v2 and v3 is performed using a rotation matrix. Next a perspective transformation is performed and four screen coordinates sxy0, sxy1, sxy2, and sxy3 are returned. An interpolation value for depth cueing on v2 to p is also returned.

v0, v1, v2, v3 -> vx, vy, vz	: (1, 15, 0)
sxy0, sxy1, sxy2, sxy3	: (1, 15, 0), (1, 15, 0)
p	: (0, 20, 12)
flag	: (0, 32, 0)

Return value

1/4 (OTZ value) average of four screen coordinate Z values.

See also

[RotAverage3\(\)](#), [RotAverageNclip4\(\)](#)

RotAverageNclip3

Perform coordinate transformation and perspective transformation for three points, and find an interpolation value, average of Z values, and outer product.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**long RotAverageNclip3(
SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,
long *sxy0, long *sxy1, long *sxy2,
long *p,
long *otz,
long *flag)**

Pointer to vectors (input)
Pointer coordinates (output)
Pointer to interpolation (output)
Pointer to OTZ value (output)
Pointer to flag (output)

Explanation

A coordinate transformation of three points *v0*, *v1*, *v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sxy0*, *sxy1*, and *sxy2* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for *v2* to *otz*.

v0, *v1*, *v2* -> *vx*, *vy*, *vz* : (1, 15, 0)
sxy0, *sxy1*, *sxy2* : (1, 15, 0), (1, 15, 0)
p : (0, 20, 12)
otz : (0, 32, 0)
flag : (0, 32, 0)

When the return value is negative, *SX*, *SY*, etc. are incorrect. When *SX* and *SY* are required, use *RotAverage3()*.

Return value

Outer product of (*sx0*, *sy0*), (*sx1*, *sy1*), (*sx2*, *sy2*).

See also

[RotAverage3\(\)](#), [RotAverageNclip4\(\)](#), [RotAverageNclip3_nom\(\)](#), [RotAverageNclipColorCol3\(\)](#), [RotAverageNclipColorDpq3\(\)](#), [RotColorDpq3\(\)](#)

RotAverageNclip3_nom

Perform coordinate transformation and perspective transformation for three points, and find an interpolation value, average of Z values, and outer product. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotAverageNclip3_nom

(**SVECTOR** *v0, **SVECTOR** *v1, **SVECTOR** *v2) Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, the interpolation value p for depth cueing corresponding to v2, an average of Z values (otz) for the three screen coordinates, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register.

The argument format and data format are as follows:

```
v0, v1, v2 -> vx, vy, vz   : (1, 15, 0)
sy0, sx0, sz0              : (1, 15, 0), (1, 15, 0), (0, 16, 0)
sx1, sy1, sz1              : (1, 15, 0), (1, 15, 0) (0, 16, 0)
sx2, sy2, sz2              : (1, 15, 0), (1, 15, 0) (0, 16, 0)
p                           : (0, 20, 12)
otz                         : (0, 32, 0)
flag                       : (0, 32, 0)
```

The operation result(s) must be retrieved from the GTE. (For further information, refer to the Inline Reference documentation.)

- (sz0,sz1,sz2) is read by macro read_sz_fifo3
- ((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro read_sxsy_fifo3
- p is read by macro read_p
- otz is read by macro read_otz
- opz is read by macro read_opz

Return value

None

See also

[RotAverage3\(\)](#), [RotAverageNclip3\(\)](#), [RotAverageNclip4\(\)](#), [RotAverageNclipColorCol3\(\)](#), [RotAverageNclipColorDpq3\(\)](#), [RotColorDpq3\(\)](#)

RotAverageNclip4

Perform a coordinate transformation and perspective transformation for four points; find an interpolation value, average of Z values, and outer product.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotAverageNclip4(
SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3,
long *sxy0, *sxy1, *sxy2, *sxy3,
long *p,

long *otz,
long *flag)

Pointer to vectors (input)
Pointer to coordinates (output)
Pointer to interpolation value (output)
Pointer to OTZ value (output)
Pointer to flag (output)

Explanation

A coordinate transformation of four points v0, v1, v2, and v3 is performed using a rotation matrix. Next a perspective transformation is performed and four screen coordinates sxy0, sxy1, sxy2 and sxy3 are returned. An interpolation value for depth cueing on v2 to p is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for v2 to otz.

v0, v1, v2, v3 -> vx, vy, vz	: (1, 15, 0)
sxy0, sxy1, sxy2, sxy3	: (1, 15, 0), (1, 15, 0)
p	: (0, 20, 12)
otz	: (0, 32, 0)
flag	: (0, 32, 0)

When the return value is negative, SX, SY, etc., are incorrect. When SX and SY are required, use RotAverage4().

Return value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2).

See also

[RotAverage4\(\)](#), [RotAverageNclip3\(\)](#)

RotAverageNclipColorCol3

Perform a coordinate transformation for three points, perspective transformation, and find a color.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotAverageNclipColorCol3(
 SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, Pointer to vectors (input)
 SVECTOR *v3, *v4, *v5, Pointer to normal vectors (input)
 CVECTOR *v6, Pointer to primary color vector (input)
 long *sxy0, *sxy1, *sxy2, Pointer to coordinate values (output)
 CVECTOR *v7, *v8, *v9, Pointer to color vectors (output)
 long *otz, Pointer to OTZ value (output)
 long *flag) Pointer to flag (output)

Explanation

A coordinate transformation of three points v0, v1, v2 is performed using a rotation matrix. Next a perspective transformation is performed and four screen coordinates sxy0, sxy1, sxy2 are returned. The remaining values are calculated as follows:

$(LLV0, LLV1, LLV2) = LLM * (v3, v4, v5)$
 $(LC0, LC1, LC2) = BK + LCM * (LLV0, LLV1, LLV2)$
 $(v7, v8, v9) = v6 * (LC0, LC1, LC2)$
 (separate multiplications)

The function also returns an average of Z values of three screen coordinates to otz.

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
v3, v4, v5 -> vx, vy, vz	: (1, 3, 12)
v6 -> r, g, b	: (0, 8, 0)
sxy0, sxy1, sxy2	: (1, 15, 0), (1, 15, 0)
v7, v8, v9 -> r, g, b	: (0, 8, 0)
otz	: (0, 32, 0)
flag	: (0, 32, 0)

When the return value is negative, SX, SY, etc., are incorrect.

Return value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2)

See also

[RotAverage3\(\)](#), [RotAverageNclip3\(\)](#), [RotAverageNclipColorCol3_nom\(\)](#), [RotAverageNclipColorDpq3\(\)](#), [RotColorDpq3\(\)](#)

RotAverageNclipColorCol3_nom

Perform a coordinate transformation for three points, perspective transformation, and find a color. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotAverageNclipColorCol3_nom(

SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2,
SVECTOR *v3, **SVECTOR** *v4, **SVECTOR** *v5,
CVECTOR *v6)

Pointer to vectors (input)

Pointer to normal vectors (input)

Pointer to primary color vector (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, an average of Z values (otz) for the three screen coordinates, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register.

$(LLV0, LLV1, LLV2) = LLM * (v3, v4, v5)$

$(LC0, LC1, LC2) = BK + LCM * (LLV0, LLV1, LLV2)$

$(v7, v8, v9) = v6 * (LC0, LC1, LC2)$

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
v3, v4, v5 -> vx, vy, vz	: (1, 3, 12)
sx0, sy0, sz0	: (1,15,0), (1,15,0), (0,16,0)
sx1, sy1, sz1	: (1,15,0), (1,15,0), (0,16,0)
sx2, sy2, sz2	: (1,15,0), (1,15,0), (0,16,0)
v6 -> r, g, b	: (0, 8, 0)
v7, v8, v9 -> r,g,b	: (0,8,0)
otz	: (0,32,0)
flag	: (0,32,0)

The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- (sz0,sz1,sz2) is read by macro read_sz_fifo3
- ((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro read_sxsy_fifo3
- ((r0,g0,b0), (r1,g1,b1), (r2,g2,b2)) is read by macro read_rgb_fifo
- p is read by macro read_p
- otz is read by macro read_otz
- opz is read by macro read_opz
- flag is returned in register v0.

Return value

None.

See also

[RotAverage3\(\)](#), [RotAverageNclip3\(\)](#), [RotAverageNclipColorCol3\(\)](#), [RotAverageNclipColorDpq3\(\)](#), [RotColorDpq3\(\)](#)

RotAverageNclipColorDpq3

Perform coordinate transformation for three points, perspective transformation, and depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long RotAverageNclipColorDpq3(
  SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,   Pointer to vectors (input)
  SVECTOR *v3, SVECTOR *v4, SVECTOR *v5,   Pointer to normal vectors (input)
  CVECTOR *v6,                             Pointer to primary color vector (input)
  long *sxy0, long *sxy1, long *sxy2,      Pointer to coordinate values (output)
  CVECTOR *v7, CVECTOR *v8, CVECTOR *v9,   Pointer to color vectors (output)
  long *otz,                               Pointer to OTZ value output)
  long *flag)                             Pointer to flag (output)
```

Explanation

A coordinate transformation of three points $v0$, $v1$, $v2$ is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates $sxy0$, $sxy1$, and $sxy2$ are returned. The function uses the interpolation value p for depth cueing; p is found by the following calculations:

$$(LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

where $v6 \times (LC0, LC1, LC2)$ indicates a separate multiplication.

The function also returns an average of the Z values of the three screen coordinates to otz .

$v0, v1, v2 \rightarrow vx, vy, vz$: (1, 15, 0)
$v3, v4, v5 \rightarrow vx, vy, vz$: (1, 3, 12)
$v6 \rightarrow r, g, b$: (0, 8, 0)
$sxy0, sxy1, sxy2$: (1, 15, 0), (1, 15, 0)
$v7, v8, v9 \rightarrow r, g, b$: (0, 8, 0)
otz	: (0, 32, 0)
$flag$: (0, 32, 0)

When the return value is negative, SX, SY, etc. are incorrect.

Return value

Outer product of $(sx0, sy0)$, $(sx1, sy1)$, $(sx2, sy2)$

See also

[RotAverage3\(\)](#), [RotAverageNclip3\(\)](#), [RotAverageNclipColorCol3\(\)](#), [RotAverageNclipColorDpq3_nom\(\)](#), [RotColorDpq3\(\)](#)

RotAverageNclipColorDpq3_nom

Perform coordinate transformation for three points, perspective transformation, and depth cueing. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotAverageNclipColorDpq3_nom(

SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, Pointer to vectors (input)
SVECTOR *v3, **SVECTOR** *v4, **SVECTOR** *v5, Pointer to normal vectors (input)
CVECTOR *v6) Pointer to primary color vector (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, an average of Z values (otz) for the three screen coordinates, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register. The interpolation value p is used in the calculation below for the desired depth cueing.

$$(LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

The argument and internal data format is as follows:

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
v3, v4, v5 -> vx, vy, vz	: (1, 3, 12)
v6 -> r, g, b	: (0, 8, 0)
sx0, sy0, sz0	: (1,15,0), (1,15,0), (0,16,0)
sx1, sy1, sz1	: (1,15,0), (1,15,0), (0,16,0)
sx2, sy2, sz2	: (1,15,0), (1,15,0), (0,16,0)
v7,v8,v9 -> r,g,b	: (0,8,0)
otz	: (0,32,0)
flag	: (0,32,0)

The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- (sz0,sz1,sz2) is read by macro read_sz_fifo3
- ((sx0,sy0),(sx1,sy1),(sx2,sy2)) is read by macro read_sxsy_fifo3
- ((r0,g0,b0), (r1,g1,b1), (r2,g2,b2)) is read by macro read_rgb_fifo
- p is read by macro read_p
- otz is read by macro read_otz
- opz is read by macro read_opz
- flag is returned in register v0.

Return value

None.

See also

[RotAverage3\(\)](#), [RotAverageNclip3\(\)](#), [RotAverageNclipColorCol3\(\)](#), [RotAverageNclipColorDpq3\(\)](#), [RotColorDpq3\(\)](#)

RotColorDpq

Perform coordinate transformation for one point, perspective transformation, and depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotColorDpq(
 SVECTOR *v0, Pointer to vector (input)
 SVECTOR *v1, Pointer to normal vector (input)
 CVECTOR *v2, Pointer to primary color vector (input)
 long *sxy, Pointer to coordinate values (output)
 CVECTOR *v3, Pointer to color vector (output)
 long *flag) Pointer to flag (output)

Explanation

A coordinate transformation for the point v0 is performed using a rotation matrix. Next a perspective transformation is performed and the screen coordinate sxy is returned. The function uses the interpolation value p for depth cueing, which is found by the following calculations:

$$LLV = LLM \times v1$$

$$LC = BK + LCM \times LLV$$

$$v3 = (1-p) \times v2 \times LC + p \times FC$$

where v2 x LC indicates a separate multiplication.

v0 -> vx, vy, vz	: (1, 15, 0)
v1 -> vx, vy, vz	: (1, 3, 12)
v2 -> r, g, b	: (0, 8, 0)
sxy	: (1, 15, 0), (1, 15, 0)
v3 -> r, g, b	: (0, 8, 0)
flag	: (0, 32, 0)

Return value

1/4 of the Z component sz of the screen coordinates.

See also

[RotColorDpq_nom\(\)](#), [RotColorDpq3\(\)](#)

RotColorDpq_nom

Perform coordinate transformation for one point, perspective transformation, and depth cueing. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

long RotColorDpq_nom(

SVECTOR *v0,	Pointer to vector (input)
SVECTOR *v1,	Pointer to normal vector (input)
CVECTOR *v2)	Pointer to primary color vector (input)

Explanation

A coordinate transformation for the point *v0* is performed using a rotation matrix. Next a perspective transformation is performed and the screen coordinates (*sx,sy,sz*) are stored in the GTE internal register. The function uses the interpolation value *p* for depth cueing and stores the obtained color vector *v3* in the internal register which is found by the following calculations:

$LLV = LLM \times v1$

$LC = BK + LCM \times LLV$

$v3 = (1-p) \times v2 \times LC + p \times FC$

where *v2* x *LC* indicates a separate multiplication.

v0 -> vx, vy, vz	: (1, 15, 0)
v1 -> vx, vy, vz	: (1, 3, 12)
sx,sy,sz	: (1,15,0), (1,15,0), (0,16,0)
v2 -> r, g, b	: (0, 8, 0)
v3 -> r, g, b	: (0, 8, 0)
flag	: (0, 32, 0)

The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- sz is read by macro read_sz2
- (sx,sy) is read by macro read_sxsy2
- p is read by macro read_p
- v3 is read by macro read_rgb2
- flag is returned in register v0.

Return value

None.

See also

RotColorDpq(), RotColorDpq3()

RotColorDpq3

Perform coordinate transformation for three points, perspective transformation, and depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotColorDpq3(
SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2,
SVECTOR *v3, **SVECTOR** *v4, **SVECTOR** *v5,
CVECTOR *v6,
long *sxy0, **long** *sxy1, **long** *sxy2,
CVECTOR *v7, **CVECTOR** *v8, **CVECTOR** *v9,
long *flag)

Pointer to vectors (input)

Pointer to normal vectors (input)

Pointer to primary color vector (input)

Pointer to coordinates (output)

Pointer to color vectors (output)

Pointer to flag (output)

Explanation

A coordinate transformation of three points v0, v1, v2 is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates sxy0, sxy1, and sxy2 are returned. The function uses the interpolation value p for depth cueing, which is found by the following calculations:

$$LLV0, LLV1, LLV2) = LLM \times (v3, v4, v5)$$

$$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$$

$$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$$

where v6 x (LC0, LC1, LC2) indicates a separate multiplication.

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
v3, v4, v5 -> vx, vy, vz	: (1, 3, 12)
v6 -> r, g, b	: (0, 8, 0)
sxy0, sxy1, sxy2	: (1, 15, 0), (1, 15, 0)
v7, v8, v9 -> r, g, b	: (0, 8, 0)
flag	: (0, 32, 0)

Return value

1/4 of the Z component sz of the screen coordinates.

See also

[RotColorDpq\(\)](#), [RotColorDpq3_nom\(\)](#)

RotColorDpq3_nom

Perform coordinate transformation for three points, perspective transformation, and depth cueing. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

long RotColorDpq3_nom

SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,

SVECTOR *v3, SVECTOR *v4, SVECTOR *v5,

CVECTOR *v6)

Pointer to vectors (input)

Pointer to normal vectors (input)

Pointer to primary color vector (input)

Explanation

A coordinate transformation of three points *v0*, *v1*, *v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sx0*, *sy0*, *sz0*, *sx1*, *sy1*, *sz1*, *sx2*, *sy2* and *sz2* are stored in the GTE internal register. The function uses the interpolation value *p* for depth cueing and stores the obtained color vector in the internal register which is found by the following calculations:

$LLV0, LLV1, LLV2 = LLM \times (v3, v4, v5)$

$(LC0, LC1, LC2) = BK + LCM \times (LLV0, LLV1, LLV2)$

$(v7, v8, v9) = (1-p) \times v6 \times (LC0, LC1, LC2) + p \times FC$

$v0, v1, v2 \rightarrow vx, vy, vz$

$v3, v4, v5 \rightarrow vx, vy, vz$

$v6 \rightarrow r, g, b$

$sxy0, sxy1, sxy2$

$v7, v8, v9 \rightarrow r, g, b$

$flag$

$: (1, 15, 0)$

$: (1, 3, 12)$

$: (0, 8, 0)$

$: (1, 15, 0), (1, 15, 0)$

$: (0, 8, 0)$

$: (0, 32, 0)$

The operation result(s) must be retrieved from the GTE (For further information, refer to the Inline Reference documentation):

- *sz0,sz1,sz2* is read by macro *read_sz_fifo3*
- *((sx0,sy0),(sx2,sy2), (sx2,sy2))* is read by macro *read_sxsy_fifo3*
- *p* is read by macro *read_p* and *v7,v8,v9* is read by macro *read_rgb_fifo*. *flag* is returned in register *v0*.

See also

[RotColorDpq\(\)](#), [RotColorDpq3\(\)](#)

RotColorMatDpq

Perform ordinate transformation, perspective transformation, and depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotColorMatDpq(
 SVECTOR *v0, Pointer to vector (input)
 SVECTOR *v1, Pointer to normal vector (input)
 SVECTOR *v2, Pointer to primary color vector (input)
 long *sxy, Pointer to coordinate values (output)
 CVECTOR *v3, Pointer to color vector (output)
 long matc, Material (input)
 long flag) Address where a flag will be stored

Explanation

A coordinate transformation for the point v0 is performed using a rotation matrix. Next a perspective transformation is performed and the coordinate sxy is returned. The function uses the interpolation value p, found by the following calculations, for depth cueing.

$$LLV = LLM \times v1$$

$$LLV = LLV^{(2^{matc})}$$

$$LC = BK + LCM \times LLV$$

$$v3 = (1-p) \times v2 \times LC + p \times FC$$

where v2 x LC indicates a separate multiplication.

v0 -> vx, vy, vz	: (1, 15, 0)
v1 -> vx, vy, vz	: (1, 3, 12)
v2 -> r, g, b	: (0, 8, 0)
sxy	: (1, 15, 0), (1, 15, 0)
v3 -> r, g, b	: (0, 8, 0)
matc	: (0, 32, 0)
flag	: (0, 32, 0)

Return value

1/4 of the Z component sz of screen coordinates.

See also

[RotColorDpq\(\)](#)

RotMatrix...

Find rotation matrix from a rotation angle.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.3	12/14/98

Syntax

```
MATRIX *RotMatrix(  
SVECTOR *r,           Input: rotation angle  
MATRIX *m)           Output: rotation matrix
```

```
MATRIX *RotMatrixXZY(SVECTOR *r, MATRIX *m)  
MATRIX *RotMatrixYXZ(SVECTOR *r, MATRIX *m)  
MATRIX *RotMatrixYZX(SVECTOR *r, MATRIX *m)  
MATRIX *RotMatrixZXY(SVECTOR *r, MATRIX *m)  
MATRIX *RotMatrixZYX(SVECTOR *r, MATRIX *m)
```

Explanation

Matrix *m* is set to a rotation matrix according to the rotation angle (*r*[0],*r*[1],*r*[2]).
A rotation angle value of 4096 is equivalent to 360 degrees. A matrix element value of 4096 is equivalent to 1.0.

When matrix is:

$$c0 = \cos(r[0]), \quad s0 = \sin(r[0])$$
$$c1 = \cos(r[1]), \quad s1 = \sin(r[1])$$
$$c2 = \cos(r[2]), \quad s2 = \sin(r[2])$$

$$mX = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix}$$

$$mY = \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix}$$

$$mZ = \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

it is the result of the following product:

Table 8-2

Function name	Matrix calculation formula	Rotation sequence
RotMatrix	mX × mY × mZ	Z axis -> Y axis -> X axis
RotMatrixXZY	mX × mZ × mY	Y axis -> Z axis -> X axis
RotMatrixYXZ	mY × mX × mZ	Z axis -> X axis -> Y axis
RotMatrixYZX	mY × mZ × mX	X axis -> Z axis -> Y axis
RotMatrixZXY	mZ × mX × mY	Y axis -> X axis -> Z axis
RotMatrixZYX	mZ × mY × mX	X axis -> Y axis -> Z axis

In GTE coordinate conversion functions such as RotTransPers(), the vector is applied from the right side. For example, with RotMatrix(), the rotation is performed in the following sequence: Z axis, Y axis, X axis.

Parameter format:

```
m->m[i][j] : (1,3,12)  
r->vx,vy,vz : (1,3,12) (where 360 degrees is 1.0)
```

Return value

m

See also

[RotMatrix_gte\(\)](#), [RotMatrixC\(\)](#), [RotMatrixX\(\)](#), [RotMatrixY\(\)](#), [RotMatrixYZ_gte\(\)](#), [RotMatrixZ\(\)](#),
[RotMatrixZYX_gte\(\)](#)

RotMatrix_gte

Find a rotation matrix from a rotation angle. Approximately 2 X faster than RotMatrix().

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.1	12/14/98

Syntax

```
MATRIX *RotMatrix_gte(  
SVECTOR *r,           Pointer to rotation angle (input)  
MATRIX *m)            Pointer to rotation matrix (output)
```

Explanation

Generates a rotation queue from the rotation angle (r[0], r[1], r[2]) in matrix m. A value of 4096 represents 360 degrees; and in matrices, 4096 represents 1.0.

The matrix is obtained by doing the following multiplication. In a coordinate conversion function such as RotTransPers() for GTE, a vector is multiplied beginning with the rightmost end. So, it is rotated around the Z-, Y-, and X-axes.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix} * \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} * \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

However,

```
c0=cos (r[0])           ,s0=sin (r[0])  
c1=cos (r[1])           ,s1=sin (r[1])  
c2=cos (r[2])           ,s2=sin (r[2])  
m -> m [i] [j]         : (1, 3, 12)  
r -> vx, vy, vz        : (1, 3, 12) (where 1.0 stands for 360  
                        degrees)
```

RotMatrix_gte() is approximately 2 X faster than RotMatrix() but the result can be different by up to 2/4096.

RotMatrix() uses the same sincos table.

Return value

m

See also

RotMatrix...(), RotMatrixC(), RotMatrixX(), RotMatrixY(), RotMatrixYXZ_gte(), RotMatrixZ(), RotMatrixZYX_gte()

RotMatrixC

Find a rotation matrix from a rotation angle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

`MATRIX *RotMatrixC(`
 `SVECTOR *r,` Pointer to rotation angle (input)
 `MATRIX *m)` Pointer to rotation matrix (output)

Explanation

Same as RotMatrix(), but results in a small table and slower speed.

Return value

m.

See also

[RotMatrix...\(\)](#), [RotMatrix_gte\(\)](#), [RotMatrixX\(\)](#), [RotMatrixY\(\)](#), [RotMatrixYXZ_gte\(\)](#), [RotMatrixZ\(\)](#), [RotMatrixZYX_gte\(\)](#)

RotMatrixX

Find a rotation matrix around the X axis.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
MATRIX *RotMatrixX(  
long r,           Rotation angle (input)  
MATRIX *m)       Pointer to rotation matrix (output)
```

Explanation

Generates a rotation queue in matrix *m* as the product of a rotation matrix around the X axis at rotation angle *r*. A value of 4096 represents a rotation angle of 360 degrees and as a matrix element, 4096 represents 1.0.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} * m$$

However,

```
c = cos (r)           , s = sin (r)  
m -> m [i] [j]       : (1, 3, 12)  
r                     : (1, 3, 12) (where 1.0 stands for 360  
                      degrees)
```

Return value

m.

See also

[RotMatrix...\(\)](#), [RotMatrix_gte\(\)](#), [RotMatrixC\(\)](#), [RotMatrixY\(\)](#), [RotMatrixYZ_gte\(\)](#), [RotMatrixZ\(\)](#), [RotMatrixZYX_gte\(\)](#)

RotMatrixY

Find a rotation matrix around the Y axis.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	3.0	12/14/98

Syntax

MATRIX *RotMatrixY(
 long *r*, Rotation angle (input)
 MATRIX *m) Pointer to rotation matrix (input/output)

Explanation

Generates a rotation queue in matrix *m* as the product of a rotation matrix around the Y axis at rotation angle *r*. A value of 4096 represents a rotation angle of 360 degrees and as a matrix element, 4096 represents 1.0.

$$\begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix} * m$$

However,

<i>c</i> = cos (<i>r</i>)	, <i>s</i> = sin (<i>r</i>)
<i>m</i> -> <i>m</i> [<i>i</i>] [<i>j</i>]	: (1, 3, 12)
<i>r</i>	: (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m

See also

[RotMatrix...\(\)](#), [RotMatrix_gte\(\)](#), [RotMatrixC\(\)](#), [RotMatrixX\(\)](#), [RotMatrixYZ_gte\(\)](#), [RotMatrixZ\(\)](#), [RotMatrixZYX_gte\(\)](#)

RotMatrixYXZ_gte

Find a rotation matrix from a rotation angle. Approximately 2 X faster than RotMatrixYXZ().

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.1	12/14/98

Syntax

MATRIX *RotMatrixYXZ_gte(
SVECTOR *r, Pointer to rotation angle (input)
MATRIX *m) Pointer to rotation matrix (output)

Explanation

Generates a rotation queue in matrix m from the rotation angle (r[0], r[1], r[2]). A value of 4096 represents a rotation angle of 360 degrees, and as a matrix element, 4096 represents 1.0.

The matrix is found by performing the following multiplication. In GTE's coordinate transformation functions (such as RotTransPers()) a vector is multiplied beginning with the rightmost end. This produces rotation around the Z axis, Y axis, and X axis.

$$\begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix} * \begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

However,

c0 = cos (r[0]),	s0 = sin (r[0])
c1 = cos (r[1]),	s1 = sin (r[1])
c2 = cos (r[2]),	s2 = sin (r[2])
m -> m [i] [j]	: (1, 3, 12)
r -> vx, vy, vz	: (1, 3, 12) (where 1.0 stands for 360 degrees)

RotMatrixYXZ_gte() is approximately 2 X faster than RotMatrixYXZ() but the result can be different (by 2/4096 or less).

RotMatrixYXZ() uses the same lookup table.

Return value

m

See also

RotMatrix...(), RotMatrix_gte(), RotMatrixC(), RotMatrixX(), RotMatrixY(), RotMatrixZ(), RotMatrixZYX_gte()

RotMatrixZ

Find a rotation matrix around the Z axis.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	3.0	12/14/98

Syntax

MATRIX *RotMatrixZ(
 long *r*, Rotation angle input
 MATRIX *m) Pointer to rotation matrix output

Explanation

Generates a rotation queue in matrix m as the product of a rotation matrix around the Z axis at rotation angle *r*. A value of 4096 represents a rotation angle of 360 degrees and as a matrix element, 4096 represents 1.0.

$$\begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} * m$$

However,

c = cos (*r*),
m -> *m* [*i*] [*j*]
r

s = sin (*r*)
: (1, 3, 12)
: (1, 3, 12) (where 1.0 stands for 360 degrees)

Return value

m.

See also

[RotMatrix...\(\)](#), [RotMatrix_gte\(\)](#), [RotMatrixC\(\)](#), [RotMatrixX\(\)](#), [RotMatrixY\(\)](#), [RotMatrixYXZ_gte\(\)](#), [RotMatrixZYX_gte\(\)](#)

RotMatrixZYX_gte

Find a rotation matrix around the z, y, and x axis. Approximately 2 X faster than RotMatrixZYX.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.1	12/14/98

Syntax

```
MATRIX *RotMatrixZYX_gte(  
SVECTOR *r,           Rotation angle (input)  
MATRIX *m)            Pointer to rotation matrix (output)
```

Explanation

Generates a rotation queue from the rotation angle (r[0], r[1], r[2]) in matrix m. A value of 4096 represents 360 degrees; and in matrices, 4096 represents 1.0.

The matrix is obtained by doing the following multiplication. In a coordinate conversion function (such as RotTransPers) for GTE, a vector is multiplied beginning with the rightmost end. So, it is rotated around the X axis, Y axis, and Z axis.

$$\begin{bmatrix} c2 & -s2 & 0 \\ s2 & c2 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c1 & 0 & s1 \\ 0 & 1 & 0 \\ -s1 & 0 & c1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & c0 & -s0 \\ 0 & s0 & c0 \end{bmatrix}$$

However,

```
c0 = cos (r[0]),           s0 = sin (r[0])  
c1 = cos (r[1]),           s1 = sin (r[1])  
c2 = cos (r[2]),           s2 = sin (r[2])  
m -> m [i] [j]            : (1, 3, 12)  
r -> vx, vy, vz           : (1, 3, 12) (where 1.0 stands for 360  
                           degrees)
```

RotMatrixZYX_gte() is approximately 2 X faster than RotMatrixZYX() but the results can be different by up to 2/4096.

RotMatrixZYX() uses the same lookup table.

Return value

m

See also

RotMatrix...(), RotMatrix_gte(), RotMatrixC(), RotMatrixX(), RotMatrixY(), RotMatrixYXZ_gte(), RotMatrixZ()

RotMeshH

Perform coordinate transformation and perspective transformation.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
void RotMeshH(  
short *Yheight,           Pointer to vertex Y coordinate (input)  
DVECTOR *Vo,              Pointer to screen coordinate (output)  
u_short *sz,              Pointer to SZ value (output)  
u_short *flag,            Pointer to flag (output)  
short Xoffset, short Zoffset, Offsets for X and Z (input)  
short m, short n,         Number of vertices (input)  
DVECTOR *base)            Pointer to base address
```

Explanation

Performs coordinate transformation and perspective transformation for the number of quadrilateral mesh vertices indicated by m x n.

Vo, sz and flag are not scalar quantities but represent m x n meshes. In other words, this function returns various vertex parameters such as DVECTOR vo[n][m], u_short sz[n][m] and u_short flag[n][m].

Arguments and internal data format are as follows:

<i>Yheight</i>	: (1, 15, 0)
<i>Vo</i> -> vx, vy	: (1, 15, 0)
<i>sz</i>	: (0, 16, 0)
<i>flag</i>	: (0, 16, 0)
<i>Xoffset, Zoffset</i>	: (1, 15, 0)
<i>m, n</i>	: (1, 15, 0)
<i>base</i>	: (1, 15, 0)

The flag must normally be set between bit 27 and bit 12 of the 32-bit flag.

See also

[RotMeshPrimQ_T\(\)](#), [RotMeshPrimR_...\(\)](#), [RotMeshPrimS_...\(\)](#)

RotMeshPrimQ_T

Two-dimensional mesh.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

void RotMeshPrimQ_T(

QMESH *msh,	Pointer to mesh model data
POLY_FT4 *prim,	Pointer to GPU packet that should be created
u_long *ot,	Pointer to ordering table
u_long otlen,	Ordering table length
long dpq,	Specifies whether depth cueing will be done (1 = yes, 0 = no)
long backc,	Specifies whether back clip will be done (1 = yes, 0 = no)
SCLIP *sclip,	Pointer to screen clip area
LINE_BUF *line_sxy)	Pointer to one line buffer for internal processing

Explanation

Perform coordinate conversion, perspective conversion, normal line clip, clipping by screen coordinates (x, y, z) and linking to OT of the following two-dimensional mesh (QMESH) data.

The H direction vertex number must be a multiple of 3 (*msh* -> *lenh* = 3 x n).

```

1-----2-----3
|       |       |
4-----5-----6
|       |       |
7-----8-----9

```

Write texture as is (fog gathers, but do not calculate light source). Set the texture coordinates.

Use the following structures. (The line buffer is secured above 1H + 3 vertices). If scratch pad is used as a line buffer, it is faster.

```

typedef struct {
    long sminX;
    long smaxS;
    long sminY;
    long smaxY;
    long sminZ;
    long smaxZ;
} SCLIP;

typedef struct {
    long sxy;
    long code;
} LINE_BUF;

```

See also

[RotMeshH\(\)](#), [RotMeshPrimR_...\(\)](#), [RotMeshPrimS_...\(\)](#)

RotMeshPrimR_...

The round mesh series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotMeshPrimR_...(
    TMESH *msh,           Pointer to mesh model data
    POLY_... *prim,       Pointer to GPU packet that should be created
    u_long *ot,           Pointer to ordering table
    u_long otlen,         Ordering table length
    long dpq,             Specifies whether depth cueing will be done (1 = yes; 0 = no)
    u_long backc)         Specifies whether back clip will be done (1 = yes; 0 = no)
```

Explanation

Perform coordinate conversion, perspective conversion, and linking to *ot* of the following round model mesh (rmesh) data.

```

2-----3
/\  /\
/  \/ \
1---0---4
```

The following round mesh functions are supported in libgte:

Table 8-3: Libgte Round Mesh Functions

Function name	Type of <i>prim</i> (2 nd arg)	Description
RotMeshPrimR_F3	POLY_F3	Perform flat shading with one vertex color (light source calculation).
RotMeshPrimR_FC3	POLY_F3	Perform complete painting with one vertex color (no light source calculation).
RotMeshPrimR_FCT3	POLY_FT3	Multiply texture with one vertex color (no light source calculation).
RotMeshPrimR_FT3	POLY_FT3	Multiply the flat-shaded items and the texture with one vertex color (light source calculation).
RotMeshPrimR_G3	POLY_G3	Perform Gouraud shading with vertex color (light source calculation).
RotMeshPrimR_GC3	POLY_G3	Perform complete Gouraud painting with vertex color (no light source calculation).
RotMeshPrimR_GCT3	POLY_GT3	Multiply the Gouraud completely painted items and the texture with vertex color (no light source calculation).
RotMeshPrimR_GT3	POLY_GT3	Multiply the Gouraud-shaded items and the texture with vertex color (light source calculation).
RotMeshPrimR_T3	POLY_FT3	Write out texture as-is.

See also

[RotMeshH\(\)](#), [RotMeshPrimQ_T\(\)](#), [RotMeshPrimS_...\(\)](#)

RotMeshPrimS_...

The strip mesh series of functions.

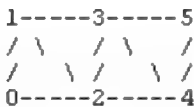
Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotMeshPrimS...(
    TMESH *msh,           Pointer to mesh model data
    POLY_... *prim,       Pointer to GPU packet that should be created
    u_long *ot,           Pointer to ordering table
    u_long otlen,         Ordering table length
    long dpq,             Specifies depth cueing (1 = yes; 0 = no)
    u_long backc)         Specifies back clip (1 = yes; 0 = no)
```

Explanation

Perform coordinate conversion, perspective conversion, and linking to *ot* of the following strip model mesh (rmesh) data.



The following strip mesh functions are supported in libgte:

Table 8-4: Libgte Strip Mesh Functions

Function name	Type of <i>prim</i> (2 nd arg)	Description
RotMeshPrimS_F3	POLY_F3	Perform flat shading with one vertex color (light source calc.).
RotMeshPrimS_FC3	POLY_F3	Perform complete painting with one vertex color (no light source calculation).
RotMeshPrimS_FCT3	POLY_FT3	Multiply texture with one vertex color (no light source calc).
RotMeshPrimS_FT3	POLY_FT3	Multiply the flat-shaded items and the texture with 1 vertex color (light source calculation).
RotMeshPrimS_G3	POLY_G3	Perform Gouraud shading with vertex color (light source calc).
RotMeshPrimS_GC3	POLY_G3	Perform complete Gouraud painting with vertex color (no light source calculation).
RotMeshPrimS_GCT3	POLY_GT3	Multiply the Gouraud completely painted items and the texture with vertex color (no light source calculation).
RotMeshPrimS_GT3	POLY_GT3	Multiply the Gouraud-shaded items and the texture with vertex color (light source calculation).
RotMeshPrimS_T3	POLY_FT3	Write out texture as-is.

See also

[RotMeshH\(\)](#), [RotMeshPrimQ_T\(\)](#), [RotMeshPrimR_...\(\)](#)

RotNclip3

Perform coordinate transformation and perspective transformation for three points, and find an interpolation value and outer product for depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long RotNclip3(  
  SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,   Pointer to vectors (input)  
  long *sxy0, long *sxy1, long *sxy2,       Pointer to coordinates (output)  
  long *p,                                   Pointer to interpolation value (output)  
  long *otz,                                 Pointer to OTZ value (output)  
  long *flag)                               Pointer to flag (output)
```

Explanation

A coordinate transformation of three points *v0*, *v1*, *v2* is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates *sx0*, *sx1*, and *sx2* are returned. An interpolation value for depth cueing on *v2* to *p* is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for *v2* to *otz*.

<i>v0</i> , <i>v1</i> , <i>v2</i> -> <i>vx</i> , <i>vy</i> , <i>vz</i>	: (1, 15, 0)
<i>sxy0</i> , <i>sxy1</i> , <i>sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>otz</i>	: (0, 32, 0)
<i>flag</i>	: (0, 32, 0)

When the return value is negative, *SX*, *SY*, etc. are incorrect. When *SX* and *SY* are needed, use *RotTransPer3()*.

Return value

Outer product of (*sx0*, *sy0*), (*sx1*, *sy1*), (*sx2*, *sy2*)

See also

[RotNclip3_nom\(\)](#), [RotNclip4\(\)](#)

RotNclip3_nom

Perform coordinate and perspective transformation for three points; get interpolation value and outer product for depth cueing. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotNclip3_nom
(SVECTOR *v0, SVECTOR *v1, SVECTOR *v2) Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, and sz2, the interpolation value p for depth cueing corresponding to v2, and an outer product value (opz) for (sx0,sy0), (sx1,sy1), and (sx2,sy2) in GTE's internal register.

v0, v1, v2 -> vx, vy, vz	: (1, 15, 0)
sx0, sy0, sz0	: (1, 15, 0), (1, 15, 0), (0,16,0)
sx1, sy1, sz1	: (1, 15, 0), (1, 15, 0), (0,16,0)
sx2,sy2, sz2	: (1, 15, 0), (1, 15, 0), (0,16,0)
p	: (0, 20, 12)
otz	: (0, 32, 0)
flag	: (0, 32, 0)

The operation result(s) must be retrieved from the GTE (for further information, refer to the Inline Reference documentation):

- (sz0,sz1,sz2) is read by macro read_sz_fifo3
- ((sx0,sy0), (sx1,sy1), (sx2,sy2)) is read by macro read_sxsy_fifo3
- p is read by macro read_p
- opz is read by macro read_opz
- flag is returned in register v0.

Return value

None.

See also

[RotNclip3\(\)](#), [RotNclip4\(\)](#)

RotNclip4

Perform coordinate transformation and perspective transformation for four points, and find an interpolation value and outer product for depth cueing.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotNclip4(
SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3,
long *sxy0, **long** *sxy1, **long** *sxy2, **long** *sxy3,
long *p,
long *otz,
long *flag)

Pointer to vectors (input)

Pointer to coordinates (output)

Pointer to interpolation value (output)

Pointer to OTZ value (output)

Pointer to flag (output)

Explanation

A coordinate transformation of four points v0, v1, v2, v3 is performed using a rotation matrix. Next a perspective transformation is performed and three screen coordinates sx0, sx1, sx2, and sx3 are returned. An interpolation value for depth cueing on v2 to p is also returned. Finally, we also receive 1/4 of the Z value of the screen coordinates for v2 to otz.

v0, v1, v2, v3 -> vx, vy, vz	: (1, 15, 0)
sxy0, sxy1, sxy2, sxy3	: (1, 15, 0), (1, 15, 0)
p	: (0, 20, 12)
otz	: (0, 32, 0)
flag	: (0, 32, 0)

When the return value is negative, SX, SY, etc. are incorrect. When SX and SY are required, use RotTransPers4().

Return Value

Outer product of (sx0, sy0), (sx1, sy1), (sx2, sy2)

See also

[RotNclip3\(\)](#)

RotPMD_...

The independent vertex PMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotPMD_...(
long *pa,           Pointer to starting address of PRIMITIVE Gp
u_long *ot,         Pointer to starting address of OT
int otlen,          Length of OT (number of bits)
int id,             Double buffer ID
int backc)          Normal line clipping ON/OFF (0: ON)
```

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-side polygons included in the independent vertex PRIMITIVE Gp, then complete the GPU packet and link it to OT.

Only polygons with an SZ value within the range [h/2, 2^16] may be linked.

The following independent vertex PMD functions are supported in libgte:

Table 8-5: Libgte Independent Vertex PMD Functions

Function Name	Description
RotPMD_F3	Flat triangle
RotPMD_F4	Flat quadrilateral
RotPMD_FT3	Flat textured triangle
RotPMD_FT4	Flat textured quadrilateral
RotPMD_G3	Gouraud triangle
RotPMD_G4	Gouraud quadrilateral
RotPMD_GT3	Gouraud textured triangle
RotPMD_GT4	Gouraud textured quadrilateral

An error may occur when placing model data (PRIMITIVEGp) on the scratch pad.

See also

[RotPMD_SV_...\(\)](#), [RotRMD_...\(\)](#), [RotRMD_SV_...\(\)](#), [RotSMD_...\(\)](#), [RotSMD_SV_...\(\)](#)

RotPMD_SV_...

The shared vertex PMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotPMD_SV_...(
long *pa,           Pointer to starting address of PRIMITIVE Gp
long *va,           Pointer to starting address of VERTEX Gp
u_long *ot,         Pointer to starting address of OT
int otlen,          Length of OT (number of bits)
int id,             Double buffer ID
int backc)          Normal line clipping ON/OFF (0: ON)
```

Explanation

These functions perform coordinate and perspective transformations on all three- and four-sided polygons included in the shared vertex PRIMITIVE Gp, then complete the GPU packet and link it to OT.

Only polygons with an SZ value within the range [h/2, 2^16] may be linked.

The following shared vertex PMD functions are supported:

Table 8-6: Libgte Independent Vertex PMD Functions

Function Name	Description
RotPMD_SV_F3	Flat triangle
RotPMD_SV_F4	Flat quadrilateral
RotPMD_SV_FT3	Flat textured triangle
RotPMD_SV_FT4	Flat textured quadrilateral
RotPMD_SV_G3	Gouraud triangle
RotPMD_SV_G4	Gouraud quadrilateral
RotPMD_SV_GT3	Gouraud textured triangle
RotPMD_SV_GT4	Gouraud textured quadrilateral

See also

[RotPMD_...\(\)](#), [RotRMD_...\(\)](#), [RotRMD_SV_...\(\)](#), [RotSMD_...\(\)](#), [RotSMD_SV_...\(\)](#)

RotRMD_...

The independent vertex RMD data series of functions.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.1	12/14/98

Syntax

```
void RotRMD...(
long *pa,           Pointer to starting address of PRIMITIVE Gp
u_long *ot,         Pointer to starting address of OT
int otlen,          Length of OT (number of bits)
int id,             Double buffer ID
int sclip,           Screen clip ON/OFF (ON=1)
int hclip,           H direction clip ([0,hclip]=display)
int vclip,           V direction clip ([0,vclip]=display)
int nclipmode)       Near Z clip mode (0=0,SCRZ/2=1)
```

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in independent vertex type PRIMITIVE Gp, then create GPU packets and link them to OT.

If *sclip* = 0, all polygons are displayed.

If *sclip* = 1, only polygons having at least one vertex that is included in the square ([0,*hclip*],[0,*vclip*]) are displayed.

If *nclipmode* = 0, polygons are far and near clipped by *sz*=[0,2^16].

If *nclipmode* = 1, polygons are far and near clipped by *sz*=[*h*,2^16] (*h*=distance of eye to screen).

No polygons are backface clipped.

The following independent vertex RMD functions are supported:

Table 8-7: Libgte Independent Vertex RMD Functions

Function Name	Description
RotRMD_F3	Flat triangle
RotRMD_F4	Flat quadrilateral
RotRMD_FT3	Flat textured triangle
RotRMD_FT4	Flat textured quadrilateral
RotRMD_G3	Gouraud triangle
RotRMD_G4	Gouraud quadrilateral
RotRMD_GT3	Gouraud textured triangle
RotRMD_GT4	Gouraud textured quadrilateral

An error may occur when placing model data (PRIMITIVEGp) on the scratch pad.

See also

[RotPMD_...\(\)](#), [RotPMD_SV_...\(\)](#), [RotRMD_SV_...\(\)](#), [RotSMD_...\(\)](#), [RotSMD_SV_...\(\)](#)

RotRMD_SV_...

The shared vertex RMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotRMD_SV_...(
long *pa,           Pointer to starting address of PRIMITIVE Gp
u_long *ot,         Pointer to starting address of OT
int otlen,          Length of OT (number of bits)
int id,             Double buffer ID
int sclip,          Screen clip ON/OFF (ON=1)
int hclip,          H direction clip ([0,hclip]=display)
int vclip,          V direction clip ([0,vclip]=display)
int nclipmode)      Near Z clip mode (0=0,SCRZ/2=1)
```

Explanation

These functions perform coordinate and perspective transformations on all three and four-sided polygons included in shared vertex type PRIMITIVE Gp, then create GPU packets and link them to OT.

If *sclip* = 0, all polygons are displayed.

If *sclip* = 1, only polygons having at least one vertex that is included in the square ([0,*hclip*],[0,*vclip*]) are displayed.

If *nclipmode* = 0, polygons are far&near clipped by $sz=[0,2^{16}]$.

If *nclipmode* = 1, polygons are far&near clipped by $sz=[h,2^{16}]$ (*h*=distance of eye to screen).

No polygons are backface clipped.

The following shared vertex RMD functions are supported:

Table 8-8: Libgte Shared Vertex RMD Functions

Function Name	Description
RotRMD_SV_F3	Flat triangle
RotRMD_SV_F4	Flat quadrilateral
RotRMD_SV_FT3	Flat textured triangle
RotRMD_SV_FT4	Flat textured quadrilateral
RotRMD_SV_G3	Gouraud triangle
RotRMD_SV_G4	Gouraud quadrilateral
RotRMD_SV_GT3	Gouraud textured triangle
RotRMD_SV_GT4	Gouraud textured quadrilateral

See also

[RotPMD_...\(\)](#), [RotPMD_SV_...\(\)](#), [RotRMD_...\(\)](#), [RotSMD_...\(\)](#), [RotSMD_SV_...\(\)](#)

RotSMD_...

The independent vertex SMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotSMD_...(
long *pa,           Pointer to starting address of PRIMITIVE Gp
u_long *ot,         Pointer to starting address of OT
int otlen,          Length of OT (number of bits)
int id,             Double buffer ID
int sclip,          Screen clip ON/OFF (ON=1)
int hclip,          H direction clip ([0,hclip]=display)
int vclip,          V direction clip ([0,vclip]=display)
int nclipmode)      Near Z clip mode (0=0,SCRZ/2=1)
```

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in independent vertex type PRIMITIVE Gp, then create GPU packets, and link them to OT.

If *sclip* = 0, all polygons are displayed.

If *sclip* = 1, only polygons with at least one vertex that is included in the square ([0,*hclip*],[0,*vclip*]) are displayed.

If *nclipmode* = 0, polygons are far&near clipped by *sz*=[0,2¹⁶].

If *nclipmode* = 1, polygons are far&near clipped by *sz*=[*h*,2¹⁶] (*h*=distance of eye to screen).

All polygons are backface clipped.

The following independent vertex SMD functions are supported in libgte:

Table 8-9: Libgte Independent Vertex SMD Functions

Function Name	Description
RotSMD_F3	Flat triangle
RotSMD_F4	Flat quadrilateral
RotSMD_FT3	Flat textured triangle
RotSMD_FT4	Flat textured quadrilateral
RotSMD_G3	Gouraud triangle
RotSMD_G4	Gouraud quadrilateral
RotSMD_GT3	Gouraud textured triangle
RotSMD_GT4	Gouraud textured quadrilateral

An error may occur when placing model data (PRIMITIVEGp) on the scratch pad.

See also

[RotPMD_...\(\)](#), [RotPMD_SV_...\(\)](#), [RotRMD_...\(\)](#), [RotRMD_SV_...\(\)](#), [RotSMD_SV_...\(\)](#)

RotSMD_SV_...

The shared vertex SMD data series of functions.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

```
void RotSMD_SV_...(
long *pa,           Pointer to starting address of PRIMITIVE Gp
u_long *ot,         Pointer to starting address of OT
int otlen,          Length of OT (number of bits)
int id,             Double buffer ID
int sclip,          Screen clip ON/OFF (ON=1)
int hclip,          H direction clip ([0,hclip]=display)
int vclip,          V direction clip ([0,vclip]=display)
int nclipmode)      Near Z clip mode (0=0,SCRZ/2=1)
```

Explanation

These functions perform coordinate transformations and perspective transformations on all three and four-sided polygons included in shared vertex type PRIMITIVE Gp, then create GPU packets, and link them to OT.

If *sclip* = 0, all polygons are displayed.

If *sclip* = 1, only polygons with at least one vertex that is included in the square ([0,*hclip*],[0,*vclip*]) are displayed.

If *nclipmode* = 0, polygons are far&near clipped by $sz=[0,2^{16}]$.

If *nclipmode* = 1, polygons are far&near clipped by $sz=[h,2^{16}]$ (*h*=distance of eye to screen).

All polygons are backface clipped.

The following shared vertex SMD functions are supported in libgte:

Table 8-10: Libgte Shared Vertex SMD Functions

Function Name	Description
RotSMD_SV_F3	Flat triangle
RotSMD_SV_F4	Flat quadrilateral
RotSMD_SV_FT3	Flat textured triangle
RotSMD_SV_FT4	Flat textured quadrilateral
RotSMD_SV_G3	Gouraud triangle
RotSMD_SV_G4	Gouraud quadrilateral
RotSMD_SV_GT3	Gouraud textured triangle
RotSMD_SV_GT4	Gouraud textured quadrilateral

See also

[RotPMD_...\(\)](#), [RotPMD_SV_...\(\)](#), [RotRMD_...\(\)](#), [RotRMD_SV_...\(\)](#), [RotSMD_...\(\)](#)

RotTrans

Perform coordinate transformation using a rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void RotTrans(  
  SVECTOR *v0,           Pointer to vector (input)  
  VECTOR *v1,            Pointer to vector (output)  
  long *flag)            Pointer to flag
```

Explanation

Calculates $v1 = RTM \times v0$.

<i>v0</i> -> vx, vy, vz	: (1, 15, 0)
<i>v1</i> -> vx, vy, vz	: (1, 31, 0)
<i>flag</i>	: (0, 32, 0)

See also

[RotTrans_nom\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#), [TransRot_32\(\)](#)

RotTrans_nom

Perform coordinate transformation using a rotation matrix. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

void RotTrans_nom (
SVECTOR *v0) Pointer to vector (input)

Explanation

Calculates $v1 = RTM \times v0$.

<i>v0</i> -> vx, vy, vz	: (1, 15, 0)
<i>v1</i> -> vx, vy, vz	: (1, 31, 0)
<i>flag</i>	: (0, 32, 0)

The operation result(s) must be retrieved from the GTE. (For further information, refer to the Inline Reference documentation.)

- (v1->vx,v1->vy,v1->vz) is read by macro read_mt
- flag is read by macro read_flag.

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#)

RotTransPers

Perform coordinate and perspective transformation for one vertex.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotTransPers(
 SVECTOR *v0, Pointer to vertex coordinate vector (input)
 long *sxy, Pointer to screen coordinates
 long *p, Pointer to interpolated value
 long *flag) Pointer to flag

Explanation

After converting the coordinate vector v0 with a rotation matrix, the function performs perspective transformation, and returns screen coordinates sx, sy. It also returns an interpolated value for depth cueing in p.

v0 -> vx, vy, vz	: (1, 15, 0)
sxy	: (1, 15, 0), (1, 15, 0)
p	: (0, 20, 12)
flag	: (0, 32, 0)

Return value

1/4 of the screen coordinate Z component sz.

See also

[RotTrans\(\)](#), [RotTransPers_nom\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#), [TransRotPers\(\)](#)

RotTransPers_nom

Perform coordinate and perspective transformation for one vertex. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotTransPers_nom(
SVECTOR *v0) Pointer to vertex coordinate vector (input)

Explanation

After converting the coordinate vector v0 with a rotation matrix, the function performs perspective transformation, and stores screen coordinates sx, sy, sz and the interpolated value p for depth cueing in the GTE internal register.

The argument and internal data format is as follows:

<i>v0</i> -> vx, vy, vz	: (1, 15, 0)
<i>sx</i>	: (1, 15, 0)
<i>sy</i>	: (1, 15, 0)
<i>sz</i>	: (0, 16, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

sz is read by macro read_sz2, (sx,sy) is read by macro read_sxsy2, p is read by macro read_p and flag is read by macro read_flag.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

Return value

None.

See also

[RotTrans\(\)](#), [RotTransPers_nom\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#)

RotTransPers3

Perform coordinate and perspective transformation of three vertices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long RotTransPers3(  
  SVECTOR *v0, SVECTOR *v1, SVECTOR *v2,    Pointer to vertex coordinate vectors  
  long *sxy0, long *sxy1, long *sxy2,        Pointer to screen coordinates  
  long *p,                                    Pointer to depth cueing interpolated value  
  long *flag)                                Pointer to flag
```

Explanation

Transforms the three coordinate vectors *v0*, *v1*, and *v2* using a rotation matrix, performs perspective transformation, and returns three screen coordinates *sxy0*, *sxy1*, and *sxy2*. It also returns to *p* an interpolated value for depth cueing corresponding to *v2*.

<i>v0</i> , <i>v1</i> , <i>v2</i> -> vx, vy, vz	: (1, 15, 0)
<i>sxy0</i> , <i>sxy1</i> , <i>sxy2</i>	: (1, 15, 0), (1, 15, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

Return value

1/4 of the screen coordinate Z component *sz* corresponding to *v2*.

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3_nom\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#), [TransRotPers3\(\)](#)

RotTransPers3_nom

Perform coordinate and perspective transformation of three vertices. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotTransPers3_nom(
SVECTOR *v0, SVECTOR *v1, SVECTOR *v2) Pointers to vertex coordinate vectors

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, and v2 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates sx0, sy0, sz0, sx1, sy1, sz1, sx2, sy2, sz2 and the interpolation value p for depth cueing corresponding to v2 in GTE's internal register.

The argument and internal data format is as follows:

<i>v0, v1, v2</i> -> vx, vy, vz	: (1, 15, 0)
<i>sx0, sy0, sz0</i>	: (1, 15, 0), (0, 16, 0)
<i>sx1, sy1, sz1</i>	: (1, 15, 0), (0, 16, 0)
<i>sx2, sy2, sz2</i>	: (1, 15, 0), (0, 16, 0)
<i>p</i>	: (0, 20, 12)
<i>flag</i>	: (0, 32, 0)

(sz0,sz1,sz2) is read by macro read_sz_fifo3, ((sx0,sy0), (sx1,sy1),(sx2,sy2) is read by macro read_sxsy_fifo3, p is read by macro read_p and flag is read by macro read_flag.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

Return value

None.

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#)

RotTransPers3N

Perform coordinate and perspective transformation for multiple triangles

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

void RotTransPers3N(

SVECTOR * <i>v0</i> ,	Pointer to vertex coordinate vector (input)
DVECTOR * <i>v1</i> ,	Pointer to vertex coordinate vector (output)
u_short * <i>sz</i> ,	Pointer to SZ value (output)
u_short * <i>flag</i> ,	Pointer to flag (output)
long <i>n</i>)	Number of triangles to process (number of input vertices/3)

Explanation

Executes RotTransPers3() for the number of triangles specified by *n*. It transforms 3 vertices at a time and stores 3 screen coordinates, an *sz* value and a *flag* value.

v0 points to an array of SVECTOR that must be 3 times *n* in length. *v1* points to an array of DVECTOR (screen coordinates) that must be 3 times *n* in length. *sz* and *flag* point to arrays of shorts that must each be *n* in length.

Arguments and internal data formats are as follows:

<i>v0</i> -> vx, vy, vz	: (1, 15, 0)
<i>v1</i> -> vx, vy	: (1, 15, 0)
<i>sz</i>	: (0, 16, 0)
<i>flag</i>	: (0, 16, 0)

Since the flag is a short, it has been right-shifted 12 places, so the information returned is that normally found between bits 27 and 12 in the 32-bit flag.

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#)

RotTransPers4

Perform coordinate and perspective transformation for 4 vertices.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

long RotTransPers4(
 SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, Pointer to vectors (input)
 SVECTOR *v3,
 long *sxy0, **long** *sxy1, **long** *sxy2, **long** *sxy3, Pointer to screen coordinates
 long *p, Pointer to interpolated value for depth cueing
 long *flag) Pointer to flag

Explanation

After transforming the four coordinate vectors v0, v1, v2, and v3 using a rotation matrix, the function performs perspective transformation, and returns four screen coordinates sxy0, sxy1, sxy2, and sxy3. It also returns an interpolation value for depth cueing to p corresponding to v3. The argument format is as follows:

v0, v1, v2, v3 -> vx, vy, vz	: (1, 15, 0)
sxy0, sxy1, sxy2, sxy3	: (1, 15, 0), (1,15,0)
p	: (0, 20, 12)
flag	: (0, 16, 0)

Return value

1/4 of the Z component sz of the screen coordinates corresponding to v3.

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4_nom\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#)

RotTransPers4_nom

Perform coordinate and perspective transformation for 4 vertices. Results obtained through GTE.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

long RotTransPers4_nom(
SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, **SVECTOR** *v3) Pointer to vectors (input)

Explanation

After performing coordinate transformation for local coordinate vectors v0, v1, v2 and v3 using a rotation matrix, this function performs perspective transformation and stores three screen coordinates (sz0), (sx1,sy1,sz1), (sx2,sy2,sz2),(sx3,sy3,sz3), and the interpolation value p for depth cueing corresponding to v3 in GTE's internal register.

The argument and internal data format is as follows:

v0,v1,v2-> vx, vy, v	: (1, 15, 0)
sx0,sy0,sz0	: (1,15,0), (1,15,0), (0,16,0)
sx1,sy1,sz1	: (1,15,0), (1,15,0), (0,16,0)
sx2,sy2,sz2	: (1,15,0), (1,15,0), (0,16,0)
sx3,sy3,sz3	: (1,15,0), (1,15,0), (0,16,0)
p	: (0,20,12)
flag	: (0,32,0)

(sz0,sz1,sz2,sz3) is read by macro read_sz_fifo4, ((sx1,sy1),(sx2,sy2),(sx3,sy3) is read by macro read_sxsy_fifo3 and p is read by macro read_p. (sx0,sy0) is returned in register v1. flag is returned in register v0.

The operation result(s) must be retrieved from the GTE.

For further information, refer to the Inline Reference documentation.

Return value

flag

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#), [RotTransSV\(\)](#)

RotTransPersN

Perform coordinate and perspective transformation.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

void RotTransPersN(
 SVECTOR *v0, Pointer to vertex coordinate vector (input)
 DVECTOR *v1, Pointer to vertex coordinate vector (output)
 u_short *sz, Pointer to SZ value (output)
 u_short *p, Pointer to interpolation value (output)
 u_short *flag, Pointer to flag (output)
 long n) Number of vertices (output)

Explanation

Executes RotTransPers() for the number of vertices specified by *n*.

The arguments and internal data formats are as follows:

<i>v0</i> -> vx, vy, vz	: (1, 15, 0)
<i>v1</i> -> vx, vy	: (1, 15, 0)
<i>sz</i>	: (0, 16, 0)
<i>flag</i>	: (0, 16, 0)

The flag must normally be set between bits 27 and 12 of the 32-bit flag.

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransSV\(\)](#)

RotTransSV

Perform coordinate translation with rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

```
void RotTransSV(  
  SVECTOR *v0,      Pointer to input: vector  
  SVECTOR *v1,      Pointer to output: vector  
  long *flag)        Pointer to output: flag
```

Explanation

RotTrans output short vector edition

```
v1 = RTM x v0  
v0->vx,vy,vz      : (1,15,0)  
v1->vx,vy,vz      : (1,15,0)  
flag               : (0,32,0)
```

See also

[RotTrans\(\)](#), [RotTransPers\(\)](#), [RotTransPers3\(\)](#), [RotTransPers3N\(\)](#), [RotTransPers4\(\)](#), [RotTransPersN\(\)](#)

rsin

Sine.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
int rsin(
int a)                Angle (in PlayStation format)
```

Explanation

Finds the sine function of the angle (in PlayStation format) ($4096 = 360 \text{ degrees} = 2\pi$) using fixed-point math (where $4096=1.0$).

a : PlayStation format ($4096 = 360 \text{ degrees} = 2\pi$)

Compared to `csin()`, `rsin()` is faster and takes up more space.

Return value

`sin (a)` : (1, 19, 12)

See also

[csin\(\)](#), [rcos\(\)](#), [ratan2\(\)](#)

ScaleMatrix

Scale a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
MATRIX *ScaleMatrix(  
MATRIX *m,           Pointer to matrix (output)  
VECTOR *v)           Pointer to scale vector (input)
```

Explanation

Scales m by v. The components of v are fixed point decimals in which 1.0 represents 4096.

If:

$$m = \begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \\ a20 & a21 & a22 \end{bmatrix}, \quad v = [sx, sy \; sz]$$

Then:

$$m = \begin{bmatrix} a00 \times sx & a01 \times sy & a02 \times sz \\ a10 \times sx & a11 \times sy & a12 \times sz \\ a20 \times sx & a21 \times sy & a22 \times sz \end{bmatrix}$$

```
m -> m [i] [j]           : (1, 19, 12)  
v -> vx, vy, vz          : (1, 19, 12)
```

Return value

m

See also

[ScaleMatrixL\(\)](#)

SetBackColor

Set back color vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void SetBackColor(
long *rbk*, long *gbk*, long *bbk*)** Colors (input)

Explanation

Sets the back color vector to (*rbk*, *gbk*, *bbk*). Color values are in the range 0 to 255.

(*rbk*, *gbk*, *bbk*) : (0, 32, 0)

See also

[SetFarColor\(\)](#)

SetColorMatrix

Set a local color matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void SetColorMatrix(  
MATRIX *m)
```

Arguments

m: Pointer to matrix (input)

Explanation

Sets a local color matrix specified by m.

$m \rightarrow m[i][j] : (1, 3, 12)$

SetFarColor

Set far color vectors.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void SetFarColor(
long *rfc*, long *gfc*, long *bfc*)** Color values (input)

Explanation

Sets the far color vector to (*rfc*, *gfc*, *bfc*). Color values are in the range 0 to 255.

(*rfc*, *gfc*, *bfc*) : (0, 32, 0)

See also

[SetBackColor\(\)](#)

SetFogFar

Set a fog parameter.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void SetFogFar(

long *a*,

Z value (0 – 65536)

long *h*)

Distance between visual point and screen

Explanation

a defines the Z value at which fog is 100%. A Z value of $0.2 * a$ will automatically make fog 0%.

a : (0, 32, 0)

h : (0, 32, 0)

See also

[SetFogNear\(\)](#), [SetFogNearFar\(\)](#)

SetFogNear

Set a fog parameter.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void SetFogNear(

long *a*,

Z value (0 – 65536 x .2)

long *h*)

Distance between visual point and screen

Explanation

a defines the Z value at which fog is 0%. A Z value of 5 x *a* will automatically make fog 100%.

a : (0, 32, 0)

h : (0, 32, 0)

See also

[SetFogFar\(\)](#), [SetFogNearFar\(\)](#)

SetFogNearFar

Set the fog parameters.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

void SetFogNearFar(
 long *a*, Z value with fog at 0% (0 – 65536)
 long *b*, Z value with fog at 100% (0 – 65536)
 long *h*) Distance between visual point and screen

Explanation

a defines the Z value with fog at 0%. *b* defines the Z value with fog at 100%.

(*b* - *a*) >= 100.

The actual value set to the DQA and DQB GTE register is calculated using the method below:

$$K = -a * b / (b - a)$$

$$c = (b << 12) / (b - a)$$

$$DQA = (K / h) << 8$$

$$DQB = c << 12$$

However, since the DQA register is 16 bit, a limit of 16 bits is set.

a : (0, 32, 0)
b : (0, 32, 0)
h : (0, 32, 0)

See also

[SetFogFar\(\)](#), [SetFogNear\(\)](#)

SetGeomOffset

Set offset values.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void SetGeomOffset(
long *ofx*, long *ofy*)** Offset input values

Explanation

Sets the offset values (*ofx*, *ofy*).

ofx, *ofy* : (1, 31, 0)

See also

[SetGeomScreen\(\)](#)

SetGeomScreen

Set the projection.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void SetGeomScreen(
long h)           Distance
```

Explanation

Sets the distance *h* (projection) from a visual point (the eye) to the screen.

h : (0, 32, 0)

See also

[SetGeomOffset\(\)](#)

SetLightMatrix

Set a local light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void SetLightMatrix(
MATRIX *m)** Pointer to matrix (input)

Explanation

Sets a local light matrix specified by *m*.

m -> m [i] [j] : (1, 3, 12)

SetMulMatrix

Multiply two matrices and set one rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

`MATRIX *SetMulMatrix(
MATRIX *m0, MATRIX *m1)` Pointer to input matrices

Explanation

Multiplies two matrices and stores that value in one constant rotation matrix.

m0, m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m0*.

See also

[SetMulRotMatrix\(\)](#)

SetMulRotMatrix

Multiply constant rotation matrix by a matrix and set one constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.6	12/14/98

Syntax

```
MATRIX *SetMulRotMatrix(  
MATRIX *m0)          Pointer to input matrix
```

Explanation

Multiplies constant rotation matrix and a matrix and stores that value in one constant rotation matrix.

m0 -> m [i] [j] : (1, 3, 12)

Return value

m0

See also

[SetMulMatrix\(\)](#)

SetRGBcd

Set primary color vector and GPU code.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void SetRGBcd(
CVECTOR *v)** Pointer to color vector and GPU code input.

Explanation

Sets the primary color vector and GPU code v.

v -> r, g, b, cd : (0, 8, 0)

SetRotMatrix

Set a constant rotation matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

**void SetRotMatrix(
[MATRIX](#) *m)** Pointer to matrix (input)

Explanation

Sets a 3x3 matrix m as a constant rotation matrix.

m -> m [i] [j] : (1, 3, 12)

See also

[SetTransMatrix\(\)](#)

SetTransMatrix

Set a constant parallel transfer vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void SetTransMatrix(
[MATRIX](#) *m) Pointer to matrix (input)

Explanation

Sets a constant parallel transfer vector specified by m.

m -> t [i] : (1, 31, 0)

See also

[SetRotMatrix\(\)](#)

Square0

Square a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

void Square0(

VECTOR *v0,

Pointer to vector (L1, L2, L3) (input)

VECTOR *v1)

Pointer to vector (L1^2, L2^2, L3^2) (output)

Explanation

Returns a vector, obtained by squaring each term of the vector *v0*, to *v1*.

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 31, 0)

Return value

Returns *v1*

See also

[Square120](#), [SquareSL00](#), [SquareSL120](#), [SquareSS00](#), [SquareSS120](#)

Square12

Square a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
void Square12(
```

VECTOR *v0,

Pointer to vector (L1, L2, L3) (input)

VECTOR *v1)

Pointer to vector ($L1^2, L2^2, L3^2$) (output)

Explanation

Returns a vector, obtained by dividing the square of each term of the vector *v0* by 4096, to *v1*.

$$v0 \rightarrow vx, vy, vz \quad : (1, 19, 12)$$

$v1 \rightarrow vx, vy, vz$: (1, 19, 12)

Return value

Returns v1

See also

Square0(), SquareSL0(), SquareSL12(), SquareSS0(), SquareSS12()

SquareRoot0

Square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long SquareRoot0(  
long a)           Value
```

Explanation

Returns the square root of a value *a*.

a : (0, 32, 0)

Return value

Returns the square root of *a*

See also

[csqrt\(\)](#), [InvSquareRoot\(\)](#), [SquareRoot12\(\)](#)

SquareRoot12

Square root.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

```
long SquareRoot12(
long a)           Value
```

Explanation

Returns the square root of a value *a*, which has (0, 20, 12) format, in (0, 20, 12) format.

a : (0, 20, 12)

Return value

Square root of *a*.

See also

[csqrt\(\)](#), [InvSquareRoot\(\)](#)

SquareSL0

Square a short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.0	12/14/98

Syntax

VECTOR *SquareSL0(

SVECTOR *v0,

VECTOR *v1)

Input: short vector (L1, L2, L3)

Output: vector (L1², L2², L3²)

Explanation

Returns a vector, obtained by squaring each term of the short vector v0, to v1.

v0 -> vx, vy, vz : (1, 15, 0)

v1 -> vx, vy, vz : (1, 31, 0)

Return value

v1

See also

[Square0\(\)](#), [Square12\(\)](#), [SquareSL12\(\)](#), [SquareSS0\(\)](#), [SquareSS12\(\)](#)

SquareSL12

Square a short vector.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	4.0	12/14/98

Syntax

VECTOR *SquareSL12(
SVECTOR *v0, Input: short vector (L1, L2, L3)
VECTOR *v1) Output: vector (L1^2, L2^2, L3^2)

Explanation

Returns a vector divided by 4096, obtained by squaring each term of the short vector v0, to v1.

v0 -> vx, vy, vz : (1, 3,12)
v1 -> vx, vy, vz : (1,19,12)

Return value

v1

See also

Square0(), Square12(), SquareSL0(), SquareSS0(), SquareSS12()

SquareSS0

Square a short vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.0	12/14/98

Syntax

SVECTOR *SquareSS0(
SVECTOR *v0, Input: short vector (L1, L2, L3)
SVECTOR *v1) Output: vector (L1^2, L2^2, L3^2)

Explanation

Returns a short vector, obtained by squaring each term of the short vector v0, to v1.

v0 -> vx,vy,vz : (1,15, 0)
v1 -> vx, vy, vz : (1,15, 0)

Return value

v1

See also

[Square0\(\)](#), [Square12\(\)](#), [SquareSL0\(\)](#), [SquareSL12\(\)](#), [SquareSS12\(\)](#)

SubPol3

Subdivide a triangle.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

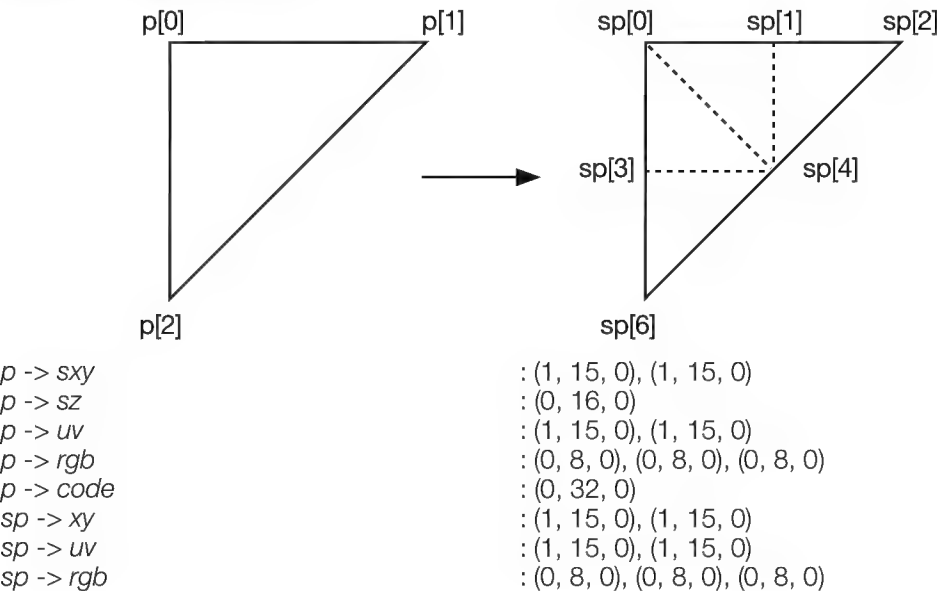
Syntax

```
void SubPol3(  
POL3 *p,           Pointer to a 3-vertex polygon  
SPOL *sp,          Pointer to subdivision vertex array  
int ndiv)           Number after subdivision  
                    0: None  
                    1: 2x2  
                    2: 4x4
```

Explanation

Subdivides a three-sided polygon *p* by the number 2^{ndiv} , and returns the subdivision vertex coordinates, texture coordinates, and RGB to a triangle in an array indicated by *sp*.

Figure 8-1 Triangle subdivision



See also

[SubPol4\(\)](#)

SubPol4

Subdivide a quadrilateral.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

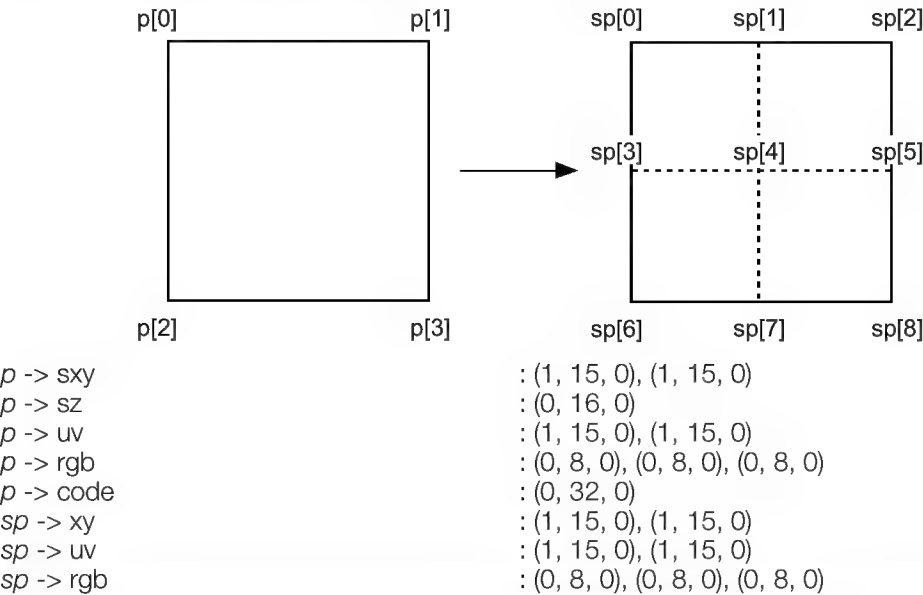
Syntax

```
void SubPol4(  
POL4 *p,           Pointer to a 4-vertex polygon  
SPOL *sp,          Pointer to subdivision vertex array  
int ndiv)           Number after subdivision  
                    0: None, 1: 2x2, 2: 4x4
```

Explanation

Subdivides a four-sided polygon *p* by the number 2^{ndiv} , and returns the subdivision vertex coordinates, texture coordinates, and RGB to an array indicated by *sp*.

Figure 8-2 Quadrilateral subdivision



See also

[SubPol3\(\)](#)

TransMatrix

Set the amount of parallel transfer.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

MATRIX

*TransMatrix(

MATRIX

*m,

VECTOR

*v)

Pointer to matrix (output)

Pointer to transfer vector (input)

Explanation

Gives an amount of parallel transfer expressed by *v* to the matrix *m*.

m -> m [i] []

: (1, 3, 12)

m -> t [i]

: (1, 31, 0)

v -> vx, vy, vz

: (1, 31, 0)

Return value

m.

See also

[TransposeMatrix\(\)](#)

TransposeMatrix

Transpose a matrix.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	2.x	12/14/98

Syntax

MATRIX *TransposeMatrix(
 MATRIX *m0, Pointer to matrix (input)
 MATRIX *m1) Pointer to matrix (output)

Explanation

Transposes matrix *m0* into *m1*.

m0 -> m [i] [j] : (1, 3, 12)
m1 -> m [i] [j] : (1, 3, 12)

Return value

Returns *m1*.

See also

[TransMatrix\(\)](#)

TransRotPers

Inversely perform rotation parallel move of RotTransPers().

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

long TransRotPers(

SVECTOR *v0,

long *sxy,

long *p,

long *flag)

Pointer to vertex coordinate vector (input)

Pointer to screen coordinate value (output)

Pointer to interpolation value for depth cueing(output)

Pointer to flag (output)

Explanation

Rotates after performing a parallel move of the coordinate vector v0 with the rotation matrix.

Performs a perspective conversion and then a coordinate conversion and returns screen coordinates sx, sy.

Also, returns the interpolation value for depth cueing to p.

v0 -> vx, vy, vz

: (1, 15, 0)

sxy

: (1, 15, 0), (1, 15, 0)

p

: (0, 20, 12)

flag

: (0, 32, 0)

Return value

1/4 of the screen coordinate Z component sz corresponding to v2.

See also

[RotTransPers\(\)](#), [TransRotPers3\(\)](#), [TransRot_32\(\)](#)

TransRotPers3

Inversely perform rotation parallel move of RotTransPers3().

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

long TransRotPers3(
 SVECTOR *v0, **SVECTOR** *v1, **SVECTOR** *v2, Pointer to vertex coordinate vector (input)
 long *sxy0, **long** *sxy1, **long** *sxy2, Pointer to screen coordinate value (output)
 long *p, Pointer to interpolation value for depth cueing (output)
 long *flag) Pointer to flag (output)

Explanation

Rotates after performing a parallel move of the three coordinate vectors v0,v1,v2 with the rotation matrix. Performs a perspective conversion and then a coordinate conversion and returns the three screen coordinates sxy0, sxy1, and sxy2.

Also, returns the interpolation value for depth cueing compatible with v2 to p.

Also, returns the screen coordinate Z item sz 1/4 compatible with v2 as the return value.

v0, v1, v2n -> vx, vy, vz	: (1, 15, 0)
sxy0, sxy1, sxy2	: (1, 15, 0), (1, 15, 0)
p	: (0, 20, 12)
flag	: (0, 32, 0)

Return value

1/4 of the screen coordinate Z component sz corresponding to v2.

See also

[TransRotPers\(\)](#), [RotTransPers3\(\)](#), [TransRot_32\(\)](#)

TransRot_32

Inversely perform rotation parallel move of RotTrans().

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.2	12/14/98

Syntax

void TransRot(

VECTOR *v0,

Pointer to vector (input)

VECTOR *v1,

Pointer to vector (output)

long *flag)

Pointer to flag (output)

Explanation

After adding the 32 bit parallel move volume to v0, performs rotation with constant rotation matrix.

v0 -> vx, vy, vz : (1, 31, 0)

v1 -> vx, vy, vz : (1, 31, 0)

flag : (0, 32, 0)

See also

[TransRotPers\(\)](#), [TransRotPers3\(\)](#), [RotTrans\(\)](#)

VectorNormal

Normalize a vector.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

```
void VectorNormal(  
VECTOR *v0,           Pointer to vector (input)  
VECTOR *v1)           Pointer to vector (output)
```

Explanation

Normalizes a vector v0 and returns the result in v1.

```
v0 -> vx, vy, vz           : (1, 31, 0)  
v1 -> vx, vy, vz           : (1, 19, 12)
```

Warning: if $((v0->vx)^2 + (v1->vx)^2 + (v2->vx)^2) > 0x7FFFFFFF$, a processor exception will occur.

Return value

Sum of squared v0 elements.

See also

[VectorNormalS\(\)](#), [VectorNormalSS\(\)](#)

VectorNormals

Normalize a vector.

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	3.0	12/14/98

Syntax

```
long VectorNormalS(
```

VECTOR *v0,	Pointer to vector (input)
SVECTOR *v1)	Pointer to vector (output)

Explanation

Normalizes a vector `v0` and returns the result in `v1`.

$$v0 \rightarrow v_x, v_y, v_z : (1, 31, 0)$$

```
v1 -> vx, vy, vz : (1, 3, 12)
```

Warning: if $((v0 \rightarrow vx)^2 + (v1 \rightarrow vx)^2 + (v2 \rightarrow vx)^2) > 0x7FFFFFFF$, a processor exception will occur.

Return value

Sum of squared v0 elements

See also

VectorNormal(), VectorNormalSS()

VectorNormalSS

Normalize a vector.

Library	Header File	Introduced	Documentation Date
libgte.lib	libgte.h	2.x	12/14/98

Syntax

```
long VectorNormalSS(  
SVECTOR *v0,           Pointer to vector (input)  
SVECTOR *v1)           Pointer to vector (output)
```

Explanation

Normalizes a vector v0 and returns the result in v1.

v0 -> vx, vy, vz : (1, 16, 0)
v1 -> vx, vy, vz : (1, 3, 12)

Warning: if $((v0->vx)^2 + (v1->vx)^2 + (v2->vx)^2) > 0x7FFFFFFF$, a processor exception will occur.

Return value

Sum of squared v0 elements

See also

VectorNormal(), VectorNormalS()

Chapter 9: Extended Graphics Library

Table of Contents

Structures

GsBG	9-3
GsBOXF	9-4
GsCELL	9-5
GsCOORD2PARAM	9-6
GsCOORDINATE2	9-7
GsDOBJ2	9-8
GsDOBJ3	9-10
GsDOBJ5	9-11
GsFOGPARAM	9-13
GsF_LIGHT	9-14
GsGLINE	9-15
GsIMAGE	9-16
GsLINE	9-17
GsMAP	9-18
GsOBJTABLE2	9-19
GsOT	9-20
GsOT_TAG	9-21
GsRVIEW2	9-22
GsSPRITE	9-23
GsVIEW2	9-25
TMD_STRUCT	9-26
_GsFCALL	9-27
_GsPOSITION	9-30

Functions

dmyGsPrst...	9-31
dmyGsTMD...	9-32
GsA4div...	9-33
GsClearOt	9-36
GsClearVcount	9-37
GsCutOt	9-38
GsDefDispBuff	9-39
GsDefDispBuff2	9-40
GsDrawOt	9-41
GsDrawOtIO	9-42
GsGetActiveBuffer	9-43
GsGetLs	9-44
GsGetLw	9-45
GsGetLws	9-46
GsGetTimInfo	9-47
GsGetVcount	9-48
GsGetWorkBase	9-49
GsInit3D	9-50
GsInitCoordinate2	9-51
GsInitFixBg16, GsInitFixBg32	9-52
GsInitGraph	9-53
GsInitGraph2	9-54
GsInitVcount	9-55

GsLinkObject3	9-56
GsLinkObject4	9-57
GsLinkObject5	9-58
GsMapModelingData	9-59
GsMulCoord0	9-60
GsMulCoord2	9-61
GsMulCoord3	9-62
GsPresetObject	9-63
GsPrst...	9-64
GsScaleScreen	9-66
GsSetAmbient	9-67
GsSetClip	9-68
GsSetClip2	9-69
GsSetClip2D	9-70
GsSetDrawBuffClip	9-71
GsSetDrawBuffOffset	9-72
GsSetFlatLight	9-73
GsSetFogParam	9-74
GsSetLightMatrix	9-75
GsSetLightMatrix2	9-76
GsSetLightMode	9-77
GsSetLsMatrix	9-78
GsSetOffset	9-79
GsSetOrign	9-80
GsSetProjection	9-81
GsSetRefView2	9-82
GsSetRefView2L	9-83
GsSetView2	9-84
GsSetWorkBase	9-85
GsSortBg, GsSortFastBg	9-86
GsSortBoxFill	9-87
GsSortClear	9-88
GsSortFixBg16, GsSortFixBg32	9-89
GsSortGLine, GsSortLine	9-90
GsSortObject3	9-91
GsSortObject4	9-92
GsSortObject4J	9-93
GsSortObject5	9-94
GsSortObject5J	9-95
GsSortOt	9-96
GsSortPoly	9-97
GsSortSprite, GsSortFastSprite, GsSortFlipSprite	9-98
GsSwapDispBuffer	9-99
GsTMDdiv...	9-100
GsTMDfast..., GsTMDfastN...	9-104
Macros	
GsClearDispArea	9-109
GsIncFrame	9-110
GsSetAzwh	9-111
External Variables	

Structures

GsBG

BG (background surface) handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsBG {
    u_long attribute;           Attribute
    short x, y;                Top left point display position
    short w, h;                BG display size
    short scrollx, scrolly;     x and y scroll values
    u_char r, g, b;            Display brightness is set in r, g, b. (Normal brightness is 128.)
    GsMAP *map;                Pointer to map data
    short mx, my;              Rotation and enlargement central point coordinates
    short scalex, scaley;      Scale values in x and y directions
    long rotate;               Rotation angle (4096 = 1 degree)
};
```

Explanation

For *attribute*, see the description in GsSPRITE.

A BG (background) is drawn as a large rectangle based on GsMAP data on a combination of small rectangles defined by GsCELL data. There is a GsBG for each BG. The BG may be manipulated via the GsBG structure.

To register a GsBG object in the ordering table, use GsSortBg().

x, y specifies the screen display position.

w, h specifies BG display size in pixels, and is not dependent on cell size or map size.

If the display area is larger than the map, the content of the map is repeatedly displayed. (Tiling function)

scrollx, scrolly specifies offset from the map display position in dots.

r, g, b specifies brightness values for red, green, and blue. The range is 0 to 255. 128 is the brightness of the original pattern; 255 doubles the brightness.

map specifies the starting address of map data with a pointer to GsMAP format map data.

mx, my specify the center of rotation and scaling as relative coordinates. The top left point of the BG is the point of origin. For example, if rotation is around the center of the BG, specify w/2 and h/2.

scalex, scaley specifies enlargement/reduction values in the x and y directions. These values are expressed in units of 4096, which stands for 1.0 (i.e. is the same size as 1.0). You can set these values up to 8 times the original size.

rotate specifies a rotation angle around the z-axis (4096 = 1 degree).

See also

[GsInitFixBg16\(\)](#), [GsSortBg\(\)](#), [GsSortFixBg16\(\)](#)

GsBOXF

Rectangle handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsBOXF {  
    u_long attribute;           Attribute (see GsLINE attributes)  
    short x, y;                Display position (top left point)  
    u_short w, h;              Size of rectangle (width, height)  
    u_char r, g, b;            Drawing color  
};
```

Explanation

GsBOXF is a structure used to draw a rectangle in a single color. To register GsBOXF in the ordering table, use GsSortBoxFill().

See also

[GsSortBoxFill\(\)](#)

GsCELL

Cells constituting BG.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsCELL {
    u_short u           Offset (X-direction) within the page
    u_short v;          Offset (Y-direction) within the page
    u_short cba;         CLUT ID
    u_short flag;        An option at the time of drawing
    u_short tpage;       Texture page number
};
```

Explanation

A rectangular array of GsCELL structures is used to describe individual cells that fit together to create a BG. Each individual GsCELL structure defines a rectangular portion of the overall BG.

cba specifies the position within the frame buffer of a CLUT corresponding to the cell. Bits 0-5 are the X position of the CLUT divided by 16. Bits 6-15 are the Y position of the CLUT.

tpage is a page number that indicates the position of a Sprite pattern within a frame buffer.

The *u* and *v* parameters specify the offset position for the sprite pattern within the texture page defined by *tpage*.

flag specifies drawing options. Bit 0 is Vertical flip (0: no flip; 1: flip). Bit 1 is Horizontal flip (0: no flip; 1: flip).

GsCOORD2PARAM

GsCOORDINATE2 parameter format.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	<i>4.0</i>	<i>12/14/98</i>

Structure

```

struct {
    VECTOR scale;           Retains coordinate scaling information
    SVECTOR rotate;         Retains coordinate rotation information
    VECTOR trans;           Retains coordinate parallel shift information
} GsCOORD2PARAM;

```

Explanation

This structure is used in order to retain information for GSCOORDINATE2 when TOD animation is used.

GsCOORDINATE2

Matrix type coordinate system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	9/1/99

Structure

```
struct GsCOORDINATE2 {
    u_long flag;           Flag indicating whether coord was rewritten
    MATRIX coord;         Matrix
    MATRIX workm;         Result of multiplication from this coordinate system to the WORLD
                           coordinate system
    GsCOORD2PARAM *param; Pointer for scale, rotation, and transfer parameters
    GsCOORDINATE2 *super; Pointer to superior coordinates
    GsCOORDINATE2 *sub;   Not in current use
};
```

Explanation

GsCOORDINATE2 has superior coordinates and is defined by the matrix type *coord*.

workm retains the result of multiplication of matrices performed by GsGetLw() and GsGetLs() in each node of GsCOORDINATE2 using the WORLD coordinates.

flag is referenced to omit calculations for a node for which calculations were already made, during GsGetLw() calculations. The external variable PSDCNT sets the flag and 0 clears it. If you change the contents of *coord*, you must clear this flag. If you neglect to clear it, GsGetLw() and GsGetLs() will fail to execute normally.

param is used for setting coord values with layout tools. It may be freely used if TOD animation is not used.

See also

[GsGetLs\(\)](#), [GsGetLw\(\)](#), [GsGetLws\(\)](#), [GsInitCoordinate2\(\)](#)

GsDOBJ2

Three-dimensional object handler

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Structure

```
struct GsDOBJ2 {
    u_long attribute;           Object attribute (32-bit)
    GsCOORDINATE2 *coord2;     Pointer to a local coordinate system
    u_long *tmd;               Pointer to model data
    u_long id;                 Reserved by the layout tool
};
```

Explanation

Used to manipulate objects in a three-dimensional model. There is a GsDOBJ2 for each object in a model.

Use GsLinkObject4() to link a GsDOBJ2 to TMD-format model data. It sets *tmd* to the starting address of the TMD model object in memory.

Use GsSortObject4() to register GsDOBJ2 in the ordering table.

coord2 is a pointer to a GsCOORDINATE2 structure defining the object's coordinate system. The location, inclination, and size of the object is defined in *coord2->coord*.

attribute is a 32-bit value containing various display attributes:

- Bits 0-2: material attenuation. (**Note:** currently not supported)

Sets the relationship between the normal gradient and brightness attenuation for light source calculations. Values range from 0 (no attenuation) to 3 (steepest attenuation). This value affects an object's material quality: for example, a steep attenuation generally produces a metallic quality. Also note that the higher the value, the longer the processing time.

Note: This parameter is invalid unless the lighting mode sets material attenuation on.

- Bits 3-5: lighting mode

Sets the light source calculation formula. Bit 5 is a switch to validate the default lighting mode set by GsSetLightMode(). The values of bits 3-4 can be:

Table 9-1: Lighting modes

Value	Operation
0	Normal (fastest) mode.
1	Fog only mode. Use GsSetFogParam() to set the fog parameter GsFOGPARAM.
2	Material attenuation only mode.
3	Applies both fog and material attenuation. Not currently supported.

- Bit 6: Light source calculation ON/OFF switch (1 = off)

Improves processing speed by eliminatin light source calculation. A texture-mapped polygon is displayed in the original texture color; an unmapped polygon is displayed in the model data color.

- Bit 7: Near clipping

If this bit is set, in cases where the polygon end point is very close to the viewpoint (distance between viewpoint and polygon < (distance between viewpoint and screen) / 2), a polygon that has overflowed during perspective transformation will not be simply clipped, but can be forcibly displayed, even if its shape is distorted.

- Bit 8: Back clipping
A polygon has a front and back determined by the order of its vertices. In the case of a convex object, it is not necessary to display the back face, so a back-facing polygon will be clipped. However, if this bit is set, a back-facing polygon can be displayed.
The current version does not support back clipping.
- Bits 9-11: Automatic division
This operation subdivides an object's component polygons at the time of execution. The number of divisions possible are: 2x2 (1), 4x4 (2), 8x8 (3), 16x16 (4), or 32x32 (5).
You can use this operation to eliminate the problems accompanying a perspective transformation, such as texture distortion and Near clipping. You must take care that memory use and processing speed are not adversely impacted as the number of divisions increase.
GsSortObject4() and GsSortObject5() are functions that create packets capable of automatic division. When using automatic division, you must pass the scratch pad address, used as a working argument, in the last argument of the packet creation function.
- Bits 28-29: Semi-transparency rate (**Note:** currently not supported)
Sets the pixel-blending formula when semi-transparency is set to ON in bit 30.

Table 9-2: Semi-transparency Rate

Value	Background	Primitive	Processing
0	0.5	0.5	Normal semi-transparency processing
1	1.0	1.0	Pixel addition
2	1.0	-1.0	50% addition
3	1.0	0.25	Pixel subtraction

- Bit 30: Semi-transparency ON/OFF
Used with the high (STP) bit of the texture color field (direct texture pattern or CLUT color field when indexed) to set semi-transparency. The semi-transparency and non-transparency of each pixel unit may be controlled using this STP bit.
- Bit 31: Display ON/OFF
When the object is not displayed, speed is improved.

See also

[GsLinkObject4\(\)](#), [GsSortObject4\(\)](#),[GsSortObject4J\(\)](#), [GsSetLightMode\(\)](#)

GsDOBJ3

Three-dimensional object handler for use with PMD format.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Structure

```
struct GsDOBJ3 {
    u_long attribute;           Object attribute (32-bit)
    GsCOORDINATE2 *coord2;     Pointer to a local coordinate system
    u_long *pmd;                Pointer to model data (PMD FORMAT)
    u_long *base;               Pointer to object base address
    u_long *sv;                 Pointer to shared vertex base address
    u_long id;                  Reserved by the layout tool
};
```

Explanation

There is a GsDOBJ3 for each object of a 3-dimensional model; GsDOBJ3 structures may be used to manipulate the 3-dimensional model.

Use GsLinkObject3() to link GsDOBJ3 to PMD file model data.

You can use GsDOBJ3 to access PMD data linked by GsLinkObject3(). Use GsSortObject3() to register GsDOBJ3 in the ordering table.

coord2 is a pointer to a coordinate system unique to an object. The location, inclination, and size of the object is reflected in a matrix set in the coordinate system to point to *coord2*.

pmd retains the starting address of PMD model data stored in memory. *pmd* is calculated and set using GsLinkObject3().

attribute is 32-bit; various display attributes are set here.

Only the attribute shown below is currently available.

- Bits 0-30: Reserved, set to zero
- Bit 31: Display ON/OFF

 This turns display ON and OFF.

id is not used unless the layout funtion is used.

See also

[GsLinkObject3\(\)](#), [GsSortObject3\(\)](#)

GsDOBJ5

Three-dimensional object handler for use with GsSortObject5().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Structure

```
struct GsDOBJ5 {
    u_long attribute;           Object attribute (32-bit)
    GsCOORDINATE2 *coord2;    Pointer to local coordinate system
    u_long *tmd;               Pointer to model data
    u_long *packet;            Pointer to preset packet area
    u_long id;                  Reserved by the layout tool
};
```

Explanation

There is a GsDOBJ5 for each object of a 3-dimensional model; GsDOBJ5 structures may be used to manipulate the 3-dimensional model.

Use GsLinkObject5() to link GsDOBJ5 to TMD file model data.

You can use GsDOBJ5 to access TMD data linked by GsLinkObject5(). Use GsSortObject5() to register GsDOBJ5 in the ordering table.

coord2 is a pointer to a coordinate system unique to an object. The location, inclination, and size of the object is reflected in a matrix set in the coordinate system to point to *coord2*.

tmd retains the starting address of TMD model data stored in memory. *tmd* is calculated and set using GsLinkObject5().

packet retains the starting address of a preset packet copied into memory. A preset packet is copied by GsPresetObject(), and is set in a GsDOBJ5 packet.

attribute is 32-bit; various display attributes are set here. An explanation of each bit follows.

- Bits 0-2: Material attenuation (not currently supported)

This sets the relationship between the normal gradient and brightness attenuation when light source calculation is performed. This takes a value of 0-3. With 0 there is no attenuation; the steepest attenuation is with 3. This parameter can be used to display an object's material quality. In general, making the attenuation steep produces a metallic quality.

Note the following points:

(a) If the material attenuation value is high, calculation takes longer and the processing requires a lot of resources.

(b) This parameter is invalid in lighting mode unless material ON is set.

- Bits 3-5: Lighting mode

This sets the light source calculation formula. It takes a value of 0-3. The values are as listed below.

- Bit 5, the highest ranking bit, is a switch to validate the lighting mode set by GsSetLightMode().

Table 9-3: Lighting Modes

Value	Operation
0	Normal mode without fog or material attenuation. This is the fastest mode and calculation takes least time.
1	Fog only mode. The fog parameter is GsFOGPARAM; set the parameter with GsSetFogParam().
2	Material attenuation only mode. The amount of attenuation is set by the material attenuation bit. Not currently supported.

Value	Operation
3	Applies both fog and material attenuation. Not currently supported.

- Bit 6: Light source calculation ON/OFF switch

This bit is used when light source calculation is not performed. When light source calculation is removed, a texture-mapped polygon is displayed in the original texture color. An unmapped polygon is displayed in the model data color.

- Bits 7-30: Reserved, set to zero.
- Bits 31: Display ON/OFF

This turns display ON and OFF.

id is not used unless the layout function is used.

See also

[GsLinkObject5\(\)](#), [GsSortObject5\(\)](#), [GsSortObject5J\(\)](#)

GsFOGPARAM

Fog (depth cueing) information.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsFOGPARAM {
    short dqa;           Parameter for the degree of merging due to depth
    long dqb;           Parameter for the degree of merging due to depth.
    u_char rfc, gfc, bfc; Background colors
};
```

Explanation

dqa and *dqb* are background color attenuation coefficients. They can be calculated using the following formulas:

$$DQA = -df * 4096/64/h$$

$$DQB = 1.25 * 4096 * 4096$$

df is the distance where the attenuation coefficient is 1; that is, the distance from the viewpoint to where the background colors are completely blended. *h* indicates a projection, or a distance from the visual point to the screen.

See also

[GsSetFogParam\(\)](#)

GsF_LIGHT

Parallel light source.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsF_LIGHT {
    int vx, vy, vz;      Directional vectors for light source
    u_char r, g, b;      Light source colors
};
```

Explanation

Holds information about a parallel light source. Use `GsSetFlatLight()` to assign the values to one of three light sources..

The light source directional vectors are specified by `vx`, `vy`, `vz`. It is unnecessary for the programmer to perform normalization, because the system does it. A polygon whose normal vectors are opposite to these directional vectors is exposed to the strongest light.

See also

[GsSetFlatLight\(\)](#)

GsGLINE

Straight line handler with gradation.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.5	12/14/98

Structure

```
struct GsGLINE {
    u_long attribute;      Attribute (see GsLINE attributes)
    short x0, y0;          Drawing start point position
    short x1, y1;          Drawing end point position
    u_char r0, g0, b0;     Drawing colors of start point
    u_char r1, g1, b1;     Drawing colors of end point
};
```

Explanation

GsGLINE is a structure used to draw straight lines with gradation. It is the same as GsLINE except that drawing colors for the starting point and end point may be specified separately.

See also

[GsSortGLine\(\)](#)

GsIMAGE

Information on image data composition.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsIMAGE {
    u_long pmode;           Pixel mode.
                             0: 4-bit CLUT
                             1: 8-bit CLUT
                             2: 16-bit DIRECT
                             3: 24-bit DIRECT
                             4: Coexistence of multiple modes
    short px, py;           Pixel data storage location within the frame buffer
    u_short pw, ph;         Pixel data width and height
    u_long *pixel;          Pointer to pixel data
    short cx, cy;           CLUT data storage location within the frame buffer
    u_short cw, ch;         CLUT data width and height
    u_long *clut;           Pointer to CLUT data
};
```

Explanation

A structure in which TIM format data information is stored by GsGetTimInfo().

See also

[GsGetTimInfo\(\)](#)

GsLINE

Straight line handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsLINE {
    u_long attribute;    Attribute (see Explanation)
    short x0, y0;        Drawing start point position
    short x1, y1;        Drawing end point position
    u_char r, g, b;      Drawing color
};
```

Explanation

GsLINE is a structure for drawing straight lines. Use GsSortLine() to register a GsLINE in the ordering table.

attribute is 32 bits, and sets various attributes for display:

- Bits 0-27: Reserved, set to 0.
- Bits 28-29: Semi-transparency rate

If semi-transparency is turned on using bit 30, bits 28 and 29 are used to set the pixel blending method.

0	50% x Back + 50% x Line
1	100% x Back + 100% x Line
2	100% x Back + 50% x Line
3	100% x Back - 100% x Line

- Bit 30: Semi-transparency ON/OFF: 1 = ON; 0 = OFF
- Bit 31: Display ON/OFF: 0 = displayed; 1 = not displayed

See also

[GsSortLine\(\)](#)

GsMAP

Map comprising BG.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsMAP {
    u_char cellw, cellh;      Cell size (0 is treated as 256.)
    u_short ncellw, ncellh;   Size of BG (in cells) (Not displayed if w or h is 0.)
    GsCELL *base;            Pointer to GsCELL structure array
    u_short *index;          Pointer to cell information
};
```

Explanation

GsMAP is map data used to compose BG from GsCELL. Map data are managed by cell index array information.

cellw, *cellh* specify the size of one cell in pixels. Note that one BG is made up of cells of the same size.

ncellw and *ncellh* set the size of the BG map in cells.

base sets the starting address of the GsCELL array.

index sets the starting address of the cell data table. Cell data is a list of index values whose size is equivalent to $(ncellw * ncellh)$ for the array specified by *base*. If a cell value is 0xFFFF it indicates a NULL (transparent) cell.

GsOBJTABLE2

Object table information.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsOBJTABLE2 {
    GsDOBJ2 *top;      Pointer to object array
    int nobj;          Number of valid objects in array
    int maxobj;         Size of object array
};
```

Explanation

When the three-dimensional animation function group is used, a three-dimensional object must be in the array in order to give effect to the object ID number specification. This array is called an object table. GsOBJTABLE2 contains information relating to the object table.

top is a pointer to the GsDOBJ2 array, within which the three-dimensional object managed by ID is created. The GsDOBJ2 array must be allocated prior to object table initialization.

maxobj is the size of array indicated by *top*; its value must be greater than the maximum value of the object handled.

nobj is the number of valid objects within the array.

GsOBJTABLE2 is initialized by GsInitObjTable2().

GsOT

Ordering table header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```

struct GsOT {
    u_long length;           Bit length of OT
    GsOT_TAG *org;           Pointer to start address of GsOT_TAG table
    u_long offset;           OT screen coordinate system Z-axis offset
    u_long point;           OT screen coordinate system Z-axis typical value
    GsOT_TAG *tag;           Pointer to current GsOT_TAG element
};

```

Explanation

The GsOT structure describes the header of the ordering table format supported by libgs. This header has pointers to the actual ordering table array, specified by the *org* and *tag* members. These members are initialized using GsClearOt().

org always points to the start of the ordering table. *tag* points to the element within the ordering table at which drawing takes place.

length sets the size of the ordering table, in values from 1-14. The actual ordering table size is $2^{**length}$ (i.e. a value of 14 indicates an array of 16384 GsOT_TAG items, while a value of 8 indicates an array of 256 GsOT_TAG items).

length and *org* values should be set first. The other members are set by GsClearOt().

GsClearOt() initializes memory from *org* through to the size indicated by *length*. Note that memory will be destroyed if the size of the GsOT_TAG array pointed to by *org* is greater than that specified by *length*.

point is used by GsSortOt() in the sorting of ordering tables.

The ordering table Z-axis offset is set by *offset*. For example, if *offset* = 256, the start of the ordering table is Z = 256. (Not yet supported.)

See also

[GsClearOt\(\)](#), [GsDrawOt\(\)](#), [GsSortOt\(\)](#), [GsCutOt\(\)](#).

GsOT_TAG

Ordering table unit.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```

struct GsOT_TAG {
    unsigned p : 24;      Pointer to next item in ordering table list
    u_char num : 8;      Number of words in current GPU packet (i.e. primitive)
};

```

Explanation

A libgs ordering table is a linked list of GsOT_TAG structures and various types of GPU primitive structures. The *p* field of a GsOT_TAG structure indicates the least significant 24-bits of a pointer to the next item in the list. A value of 0xFFFFFFFF indicates the end of the list.

The [GsOT](#) structure is used by libgs to manage an array of GsOT_TAG items. Allocate an array of GsOT_TAG structures after initializing your GsOT structure.

GsRVIEW2

View handler (Reference type).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsRVIEW2 {
    long vpx, vpy, vpz;           Viewpoint coordinates
    long vrz, vry, vrz;           Reference point coordinates
    long rz;                      Viewpoint twist
    GsCOORDINATE2 *super;        Pointer to the coordinate system that sets the viewpoint
};
```

Explanation

GsRVIEW2 holds viewpoint information, and is set in libgs by GsSetRefView2(). *vpx*, *vpy*, *vpz* are the viewpoint coordinates in the coordinate system displayed by *super*.

vrz, *vry*, *vrz* are the reference point coordinates in the coordinate system displayed by *super*.

When the z axis is a vector from the viewpoint to the reference point, *rz* specifies the screen inclination against the z axis in fixed decimal format, with 4096 equivalent to one degree.

Viewpoint and reference point coordinate systems are set in *super*. As an example of using this function, an airplane cockpit view can be realized simply by setting *super* to the airplane coordinate system.

See also

[GsSetRefView2\(\)](#), [GsSetRefView2L\(\)](#)

GsSPRITE

Sprite handler.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsSPRITE {
    u_long attribute;    32 bits (see Explanation)
    short x, y;          Screen display position of the top left point
    u_short w, h;        Width and height of Sprite in pixels (Not displayed if w or h is 0.)
    u_short tpage;       Sprite pattern texture page number (0-31)
    u_char u, v;          Sprite pattern offset within the page from top left point. Range is ((0, 0) -
                        (255, 255).
    short cx, cy;        Starting position of CLUT in VRAM. (Valid for 4-bit/8-bit mode only)
    u_char r, g, b;       Red, green, blue brightness values (0-255; original brightness is 128.)
    short mx, my;        Rotation and enlargement central point coordinates
    short scalex, scaley; Scaling values in x and y directions (4096 = 1.0); can be up to 8 times
                        original size
    long rotate;         Sets rotation around the z-axis in fixed-decimal format (4096 = 1 degree)
};
```

Explanation

A structure used to handle a Sprite. Makes it possible to manipulate each Sprite via its parameters.

To register a GsSPRITE in the ordering table, use GsSortFlipSprite(), GsSortSprite(), or GsSortFastSprite().

mx, *my* specify the coordinates used as the center of rotation and scaling. For example, if rotation is desired around the center of the Sprite, specify *w/2* and *h/2* as *mx* and *my*.

attribute is 32 bits, and sets various attributes for display. An explanation of each bit follows.

- Bits 0-5: Reserved, set to zero.
- Bit 6: Brightness adjustment ON/OFF switch
This bit sets Sprite pattern pixel colors according to (*r*, *g*, *b*) values. If this bit is set to 1, brightness is not adjusted, and (*r*, *g*, *b*) values are ignored.
- Bits 7-21: Reserved, set to zero.
- Bits 22-23: Vertical flipping, horizontal flipping
0 = not flipped; 1 = flipped
- Bits 24-25: Color mode
A Sprite pattern has 4-bit mode and 8-bit mode, both of which use the color table, and 15-bit mode, which directly displays colors. These bits are used to select any of these modes.
0 = 4-bit CLUT; 1 = 8-bit CLUT; 2 = 15-bit direct.
- Bit 26: Reserved, set to zero.
- Bit 27: Rotation enlargement/reduction function
This bit turns on or off the Sprite enlargement function (0 = on, 1 = off). If rotation or enlargement of the Sprite is not needed, this bit should be set to OFF for high speed processing.
GsSortFastSprite() and GsSortFlipSprite() ignore this bit and always set the enlargement function to off.
- Bits 28-29: Semi-transparency rate
When semi-transparency is set to ON with bit 30, the semi-transparency rate sets the pixel-blending formula:

Table 9-4: Semi-transparency Rate

0	Normal semi-transparency processing	50% x Back + 50% x Sprite
1	Pixel addition	100% x Back + 100% x Sprite
2	50% addition	100% x Back + 50% x Sprite
3	Pixel subtraction	100% x Back - 100% x Sprite

- Bit 30: Semi-transparency ON/OFF (1 = on, 0 = off)
This bit must be used with the uppermost bit (STP bit) of the texture color field (texture pattern when direct and CLUT color field when indexed) to set semi-transparency,. Also, the semi-transparency and non-transparency of each pixel unit may be controlled using this STP bit.
- Bit 31: Display ON/OFF (0 = displayed, 1 = not displayed)
This turns display ON and OFF.

SPRT primitives are used when neither rotation, enlargement, nor reduction are performed by GsSortSprite(). Consequently, it is necessary to keep track of whether the uv, wh of a texture is odd or even.

GsSortFlipSprite() uses POLY_FT4 unconditionally.

With GsSortSprite(), *w*, *h* are set to 1 less than the desired dimensions, to handle the lower-right texture page problem. Consequently, the texture is displayed slightly enlarged. (When programming, the lower-right line can be included, since reduction takes place within the function.)

Since GsSortFlipSprite() displays POLY_FT4 at the original scale, the rendering rules dictate that the lower right corner cannot be used. When flipping, to ensure proper display, the lower left line should not be used either.

See also

[GsSortFlipSprite\(\)](#), [GsSortSprite\(\)](#), [GsSortFastSprite\(\)](#)

GsVIEW2

View handler (matrix type).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Structure

```
struct GsVIEW2 {
    MATRIX view;           Matrix used to change from superior coordinates to viewpoint
                           coordinates
    GsCOORDINATE *super;   Pointer to the coordinate system that sets viewpoint
};
```

Explanation

Sets the viewpoint coordinate system, and specifies the matrix used by view to change from superior coordinates to viewpoint coordinates.

The function that sets GsVIEW2 is GsSetView2().

See also

[GsSetView2\(\)](#)

TMD_STRUCT

TMD data object header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	<i>4.0</i>	<i>12/14/98</i>

Structure

```
typedef struct {
    u_long *vertop;    VERTEX start address
    u_long vern;       VERTEX coefficient
    u_long *nortop;    NORMAL start address
    u_long norm;       NORMAL coefficient
    u_long *primtop;   PRIMITIVE start address
    u_long primn;      PRIMITIVE coefficient
    u_long scale;      Scaling factor
} TMD_STRUCT;
```

Explanation

A structure in the OBJ TABLE section within the TMD data. It contains information regarding the pointer which displays where each object is stored.

_GsFCALL

The function table for GsSortObject4J() and GsSortObject5J().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Structure

```
struct _GsFCALL {
    PACKET (*f3[2][3])(), (*nf3[2])(), (*g3[2][3])(), (*ng3[2])();
    PACKET (*tf3[2][3])(), (*ntf3[2])(), (*tg3[2][3])(), (*ntg3[2])();
    PACKET (*f4[2][3])(), (*nf4[2])(), (*g4[2][3])(), (*ng4[2])();
    PACKET (*tf4[2][3])(), (*ntf4[2])(), (*tg4[2][3])(), (*ntg4[2])();
    PACKET (*f3g[3])(), (*g3g[3])();
    PACKET (*f4g[3])(), (*g4g[3])();
};
```

Members

Each member is a pointer to a low-level function.

<i>f3, g3, tf3, tg3, f4, g4, tf4, tg4</i>	Pointer to polygon types
First matrix:	Division/no division
GsDivMODE_DIV/GsDivMode_NDIV	
Second matrix:	Light source calculation mode
GsLMODE_NORMAL/GsLMODE_FOG/ GsLMODE_LOFF	
<i>nf3, ng3, ntf3, ntg3, nf4, ng4, ntf4, ntg4</i>	Pointer to polygon types
First matrix:	Division/no division
GsDivMODE_DIV/GsDivMode_NDIV	
<i>f3g, g3g, f4g, g4g</i>	Gradation polygon type
First array:	Light source calculation mode
GsLMODE_NORMAL/GsLMODE_FOG/ GsLMODE_LOFF	

Explanation

GsSortObject4() and GsSortObject5() dispatch attributes, pre-set data, etc. and call low-level functions. There are 64 low-level functions, and a single application is unlikely to use all of them.

With GsSortObject4J() and GsSortObject5J(), you don't need to link with unnecessary low-level functions, thereby making the code more compact. These functions are compatible with GsSortObject4() and GsSortObject5(), which organize low-level functions as tables.

_GsFCALL is the structure in which the function table is defined. The function table is organized according to polygon type, whether or not division is performed, and light-source calculation mode.

The relevant functions are linked by entering the pointers of the appropriate low-level functions in each of the elements. It is possible to avoid linking by not including the pointers and not making extern declarations. However, if a function that does not have a pointer is called, a BUS ERROR is generated.

The example below shows the use of GsSortObject5J() with appropriate functions in all the elements. In this example, GsSortObject5J() functions the same as GsSortObject5(). This example is included in comments in the file libgs.h.

```
/* extern and fook only using functions */
extern _GsFCALL GsFCALL5; /* GsSortObject5J Func Table */
jt_init() /* Gs SortObject5J Fook Func */
{
    PACKET *GsPrstF3NL(), *GsPrstF3LFG(), *GsPrstF3L(), *GsPrstNF3();
    PACKET *GsTMDdivF3NL(), *GsTMDdivF3LFG(), *GsTMDdivF3L(), *GsTMDdivNF3();
    PACKET *GsPrstG3NL(), *GsPrstG3LFG(), *GsPrstG3L(), *GsPrstNG3();
```

```

PACKET *GsTMDdivG3NL(), *GsTMDdivG3LFG(), *GsTMDdivG3L(), *GsTMDdivNG3();
PACKET *GsPrstTF3NL(), *GsPrstTF3LFG(), *GsPrstTF3L(), *GsPrstTNF3();
PACKET *GsTMDdivTF3NL(), *GsTMDdivTF3LFG(), *GsTMDdivTF3L(), *GsTMDdivTNF3();
PACKET *GsPrstTG3NL(), *GsPrstTG3LFG(), *GsPrstTG3L(), *GsPrstTNG3();
PACKET *GsTMDdivTG3NL(), *GsTMDdivTG3LFG(), *GsTMDdivTG3L(), *GsTMDdivTNG3();
PACKET *GsPrstF4NL(), *GsPrstF4LFG(), *GsPrstF4L(), *GsPrstNF4();
PACKET *GsTMDdivF4NL(), *GsTMDdivF4LFG(), *GsTMDdivF4L(), *GsTMDdivNF4();
PACKET *GsPrstG4NL(), *GsPrstG4LFG(), *GsPrstG4L(), *GsPrstNG4();
PACKET *GsTMDdivG4NL(), *GsTMDdivG4LFG(), *GsTMDdivG4L(), *GsTMDdivNG4();
PACKET *GsPrstTF4NL(), *GsPrstTF4LFG(), *GsPrstTF4L(), *GsPrstTNF4();
PACKET *GsTMDdivTF4NL(), *GsTMDdivTF4LFG(), *GsTMDdivTF4L(), *GsTMDdivTNF4();
PACKET *GsPrstTG4NL(), *GsPrstTG4LFG(), *GsPrstTG4L(), *GsPrstTNG4();
PACKET *GsTMDdivTG4NL(), *GsTMDdivTG4LFG(), *GsTMDdivTG4L(), *GsTMDdivTNG4();
PACKET *GsPrstF3GNL(), *GsPrstF3GLFG(), *GsPrstF3GL();
PACKET *GsPrstG3GNL(), *GsPrstG3GLFG(), *GsPrstG3GL();

/* flat triangle */
GsFCALL5.f3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstF3L;
GsFCALL5.f3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstF3LFG;
GsFCALL5.f3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstF3NL;
GsFCALL5.f3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivF3L;
GsFCALL5.f3[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivF3LFG;
GsFCALL5.f3[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivF3NL;
GsFCALL5.nf3[GsDivMODE_NDIV]                 = GsPrstNF3;
GsFCALL5.nf3[GsDivMODE_DIV]                 = GsTMDdivNF3;
/* gour triangle */
GsFCALL5.g3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstG3L;
GsFCALL5.g3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstG3LFG;
GsFCALL5.g3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstG3NL;
GsFCALL5.g3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivG3L;
GsFCALL5.g3[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivG3LFG;
GsFCALL5.g3[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivG3NL;
GsFCALL5.ng3[GsDivMODE_NDIV]                 = GsPrstNG3;
GsFCALL5.ng3[GsDivMODE_DIV]                 = GsTMDdivNG3;
/* texture flat triangle */
GsFCALL5.tf3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTF3L;
GsFCALL5.tf3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstTF3LFG;
GsFCALL5.tf3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstTF3NL;
GsFCALL5.tf3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivTF3L;
GsFCALL5.tf3[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivTF3LFG;
GsFCALL5.tf3[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivTF3NL;
GsFCALL5.ntf3[GsDivMODE_NDIV]                 = GsPrstTNF3;
GsFCALL5.ntf3[GsDivMODE_DIV]                 = GsTMDdivTNF3;
/* texture gour triangle */
GsFCALL5.tg3[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTG3L;
GsFCALL5.tg3[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstTG3LFG;
GsFCALL5.tg3[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstTG3NL;
GsFCALL5.tg3[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivTG3L;
GsFCALL5.tg3[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivTG3LFG;
GsFCALL5.tg3[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivTG3NL;
GsFCALL5.ntg3[GsDivMODE_NDIV]                 = GsPrstTNG3;
GsFCALL5.ntg3[GsDivMODE_DIV]                 = GsTMDdivTNG3;
/* flat quad */
GsFCALL5.f4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstF4L;
GsFCALL5.f4[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstF4LFG;
GsFCALL5.f4[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstF4NL;
GsFCALL5.f4[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivF4L;
GsFCALL5.f4[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivF4LFG;
GsFCALL5.f4[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivF4NL;
GsFCALL5.nf4[GsDivMODE_NDIV]                 = GsPrstNF4;
GsFCALL5.nf4[GsDivMODE_DIV]                 = GsTMDdivNF4;
/* gour quad */

```

```

GsFCALL5.g4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstG4L;
GsFCALL5.g4[GsDivMODE_NDIV][GsLMODE_FOG]     = GsPrstG4LFG;
GsFCALL5.g4[GsDivMODE_NDIV][GsLMODE_LOFF]    = GsPrstG4NL;
GsFCALL5.g4[GsDivMODE_DIV][GsLMODE_NORMAL]   = GsTMDdivG4L;
GsFCALL5.g4[GsDivMODE_DIV][GsLMODE_FOG]      = GsTMDdivG4LFG;
GsFCALL5.g4[GsDivMODE_DIV][GsLMODE_LOFF]     = GsTMDdivG4NL;
GsFCALL5.ng4[GsDivMODE_NDIV]                  = GsPrstNG4;
GsFCALL5.ng4[GsDivMODE_DIV]                   = GsTMDdivNG4;
/* texture flat quad */
GsFCALL5.tf4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTF4L;
GsFCALL5.tf4[GsDivMODE_NDIV][GsLMODE_FOG]    = GsPrstTF4LFG;
GsFCALL5.tf4[GsDivMODE_NDIV][GsLMODE_LOFF]   = GsPrstTF4NL;
GsFCALL5.tf4[GsDivMODE_DIV][GsLMODE_NORMAL]  = GsTMDdivTF4L;
GsFCALL5.tf4[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivTF4LFG;
GsFCALL5.tf4[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivTF4NL;
GsFCALL5.ntf4[GsDivMODE_NDIV]                = GsPrstTNF4;
GsFCALL5.ntf4[GsDivMODE_DIV]                 = GsTMDdivTNF4;
/* texture gour quad */
GsFCALL5.tg4[GsDivMODE_NDIV][GsLMODE_NORMAL] = GsPrstTG4L;
GsFCALL5.tg4[GsDivMODE_NDIV][GsLMODE_FOG]    = GsPrstTG4LFG;
GsFCALL5.tg4[GsDivMODE_NDIV][GsLMODE_LOFF]   = GsPrstTG4NL;
GsFCALL5.tg4[GsDivMODE_DIV][GsLMODE_NORMAL]  = GsTMDdivTG4L;
GsFCALL5.tg4[GsDivMODE_DIV][GsLMODE_FOG]     = GsTMDdivTG4LFG;
GsFCALL5.tg4[GsDivMODE_DIV][GsLMODE_LOFF]    = GsTMDdivTG4NL;
GsFCALL5.ntg4[GsDivMODE_NDIV]                = GsPrstTNG4;
GsFCALL5.ntg4[GsDivMODE_DIV]                 = GsTMDdivTNG4;
/* gradation triangle */
GsFCALL5.f3g[GsLMODE_NORMAL]                 = GsPrstF3GL;
GsFCALL5.f3g[GsLMODE_FOG]                   = GsPrstF3GLFG;
GsFCALL5.f3g[GsLMODE_LOFF]                  = GsPrstF3GNL;
GsFCALL5.g3g[GsLMODE_NORMAL]                 = GsPrstG3GL;
GsFCALL5.g3g[GsLMODE_FOG]                   = GsPrstG3GLFG;
GsFCALL5.g3g[GsLMODE_LOFF]                  = GsPrstG3GNL;
}

```

See also

[GsSortObject4J\(\)](#), [GsSortObject5J\(\)](#).

_GsPOSITION

Two-dimensional offset.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	<i>4.0</i>	<i>12/14/98</i>

Structure

```

struct _GsPOSITION {
    short offx;           Rendering offset (x direction)
    short offy;           Rendering offset (y direction)
};

```

Explanation

Two-dimensional rendering offset.

Functions

dmyGsPrst...

Jump Table dummy function group (PMD)

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

PACKET *dmyGsPrst... (*void*)

Explanation

When this function is called for the first time, the jump table entry name is printed to standard output. It is used as a low-level dummy function and is used when distinguishing which entry is being called.

For debugging use.

Return value

Pointer to the packet.

dmyGsTMD...

Jump Table dummy function group (TMD).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

PACKET *dmyGsTMD... (*void*)

Explanation

When this function is called for the first time, the jump table entry name is printed in standard output. It is used as a low-level function dummy and is utilized when distinguishing which entry is being called.

For debugging use.

Return value

Pointer to the packet.

GsA4div...

Low-level functions for GsSortObject4J() (performs automatic division).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	12/14/98

Syntax

```

PACKET *GsA4div...(
TMD_P_... *op,           Pointer to starting address of TMD data primitives
VERT *vp,                 Pointer to starting address of TMD data vertices TMD
VERT *np,                 Pointer to starting address of TMD data normals
PACKET *pk,              Pointer to top address of GPU packet buffer
int n,                     Number of primitives
int shift,                 OT shift bit
GsOT *ot,                 Pointer to GsOT
u_long *scratch)          Pointer to starting address of unused scratch pad

```

```

PACKET *GsA4divN...(
TMD_P_... *op,           Pointer to starting address of TMD data primitives
VERT *vp,                 Pointer to starting address of TMD data vertices TMD
PACKET *pk,              Pointer to top address of GPU packet buffer
int n,                     Number of primitives
int shift,                 OT shift bit
GsOT *ot,                 Pointer to GsOT
u_long *scratch)          Pointer to starting address of unused scratch pad

```

Explanation

Performs active automatic division based on Z-values, polygon size, etc.

To use these functions, they must be registered in GsFCALL4 as low-order functions, and the number of divisions must be specified in the attributes of GsDOBJ4.

Because each of these functions uses a relatively large amount of code, it would be more efficient to use only the code needed for the polygon types used.

Parameters for division include Z-values, polygon size, and GTE calculation overflow flags. These are set using the GsSetAzwh (az, aw, ah) macro.

The active division algorithm is as follows:

1. Do not divide polygons that are further away than az and that do not cause overflow in GTE calculations.
2. If cases other than 1, perform divisions (go to step 3).
3. If polygon size does not exceed aw, ah, and there is no overflow in GTE calculations, then halt division there.

Otherwise, reduce by 1/2 in the x and y directions, and divide into four sections. Call step 3 recursively. If the maximum value for divisions (the number of divisions in attribute) is reached, then halt division.

For function types which do not operate on normals within the data (e.g. GsA4divN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsA4div...),

Low-level functions in libgs that support automatic division are shown below.

Table 9-5: GsA4div...() [have normals]

Low-level function name	First arg (op) type	Description
GsA4divF3L	TMD_P_F3	Flat triangle (light source calculation)
GsA4divF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)

Low-level function name	First arg (op) type	Description
GsA4divF3NL	TMD_P_F3	Flat triangle
GsA4divF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsA4divF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsA4divF4NL	TMD_P_F4	Flag quadrilateral
GsA4divG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsA4divG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsA4divG3NL	TMD_P_G3	Gouraud triangle
GsA4divG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsA4divG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsA4divG4NL	TMD_P_G4	Gouraud quadrilateral
GsA4divTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsA4divTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsA4divTF3NL	TMD_P_TF3	Textured flat triangle
GsA4divTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsA4divTF4LFG	TMD_P_TF4	Flat quadrilateral (light source calculation +FOG)
GsA4divTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsA4divTF4LM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +mip-map)
GsA4divTF4LFGM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG+mip-map)
GsA4divTF4NLM	TMD_P_TF4	Textured flat quadrilateral (mip-map)
GsA4divTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsA4divTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsA4divTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsA4divTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsA4divTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsA4divTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsA4divTG4LM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +mip-map)
GsA4divTG4LFGM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation+FOG+mip-map)
GsA4divTG4NLM	TMD_P_TG4	Textured Gouraud quadrilateral (mip-map)

Table 9-6: GsA4divN...() [no normals]

Low-level function name	First arg (op) type	Description
GsA4divNF3	TMD_P_NF3	Flat triangle
GsA4divNF4	TMD_P_NF4	Flat quadrilateral
GsA4divNG3	TMD_P_NG3	Gouraud triangle
GsA4divNG4	TMD_P_NG4	Gouraud quadrilateral
GsA4divTNF3	TMD_P_TNF3	Textured flat triangle
GsA4divTNF4	TMD_P_TNF4	Textured flat quadrilateral

Low-level function name	First arg (op) type	Description
GsA4divTNF4M	TMD_P_TNF4	Textured flat quadrilateral (mip-map)
GsA4divTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsA4divTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral
GsA4divTNG4M	TMD_P_TNG4	Textured Gouraud quadrilateral (mip-map)

GsTMDdiv functions must be registered in GsFCALL4 when using the conventional fixed division method.

Return value

Starting address of unused packet area.

See also

[GsTMDdiv...\(\)](#), [GsSetAzwh\(\)](#), [GsSortObject4J\(\)](#).

GsClearOt

Initialize a libgs ordering table structure.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsClearOt(
  u_short offset,      Ordering table offset value
  u_short point,       Ordering table average Z value
  GsOT *otp)           Pointer to ordering table
```

Explanation

Initializes the libgs-style ordering table specified by the *otp* parameter. The *length* field of the GsOT structure must be properly set before this function is called. *offset* specifies the Z-depth value used for the start of the ordering table. *point* represents the average Z-depth of the entire ordering table and is used to determine depth priority when linking multiple ordering tables together.

See also

[GsDrawOt\(\)](#), [GsCutOt\(\)](#), [GsSortOt\(\)](#)

GsClearVcount

Clear vertical retrace counter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

void GsClearVcount(void)

Explanation

Clears the vertical retrace counter.

See also

[GsGetVcount\(\)](#), [GsInitVcount\(\)](#)

GsCutOt

OT separation.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax**GsOT *GsCutOt(**

GsOT *<i>ot_src</i>,	Pointer to old OT
GsOT *<i>ot_dest</i>)	Pointer to new OT

Explanation

Moves the drawing commands registered in the *ot_src* ordering table to the *ot_dest* ordering table. The *length* and *tag* fields of *ot_src* are reset to zero. The *tag* field of *ot_dest* is updated to point at the drawing command which was at the start of *ot_src*. Afterwards, *ot_dest* can be used to access the ordering table.

Return value

ot_dest starting address.

See also

[GsClearOt\(\)](#), [GsDrawOt\(\)](#), [GsSortOt\(\)](#)

GsDefDispBuff

Define double buffers.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsDefDispBuff (
  u_short x0, u_short y0,      Buffer 0 origin point coordinates (top left point)
  u_short x1, u_short y1)     Buffer 1 origin point coordinates (top left point)
```

Explanation

Defines the display areas used for double-buffering.

x0 and *y0* specify the frame buffer coordinates for buffer #0. *x1* and *y1* specify the frame buffer coordinates for buffer #1. Normally, buffer #0 is located at (0,0) and buffer #1 is located at (0, *yres*), where *yres* is the vertical resolution specified using `GsInitGraph()`.

If *x0*, *y0* and *x1*, *y1* are specified as the same coordinates, the double buffers are released. However, double-buffer swapping of even-numbered and odd-numbered fields is performed automatically when *x0*, *y0* and *x1*, *y1* are specified as the same coordinates in interlace mode.

`GsSwapDispBuffer()` is used to swap double buffers. The double buffer is implemented by the GPU or GTE offset. Set the `libgpu` or `libgte` offset with `GsInitGraph()`. When using the `libgpu` offset, coordinate values based on the coordinate system using the upper left point in the double buffer as the origin are created in the packet (add the offset at the time of drawing, not at the time of packet preparation).

See also

[GsInitGraph\(\)](#), [GsDefDispBuff2\(\)](#), [GsSwapDispBuffer\(\)](#), [GsGetActiveBuffer\(\)](#)

GsDefDispBuff2

Define double buffers.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsDefDispBuff2(
u_short x0, u_short y0,      Buffer 0 origin point coordinates (top left point)
u_short x1, u_short y1)    Buffer 1 origin point coordinates (top left point)
```

Explanation

Defines the double buffer. Differs from GsDefDispBuff() only in the modification of internal variables. These modifications are not updated in libgpu and libgte until GsSwapDispBuff() is called; for immediate update, call GsDefDispBuff() instead.

Settings can be changed in the middle of the program without affecting the screen.

See also

[GsDefDispBuff\(\)](#), [GsSwapDispBuffer\(\)](#), [GsGetActiveBuffer\(\)](#)

GsDrawOt

Process GPU commands registered to OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsDrawOt(  
  GsOT \*ot           Pointer to OT
```

Explanation

Starts execution of commands registered in OT, specified by *ot*. Because processing is performed in the background, GsDrawOt() returns immediately.

See also

[GsClearOt\(\)](#), [GsCutOt\(\)](#), [GsSortOt\(\)](#), [GsDrawOtIO](#)

GsDrawOtIO

Process GPU commands (I/O version) allocated to OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.5	12/14/98

Syntax

```
void GsDrawOtIO(  
  GsOT \*ot          Pointer to OT
```

Explanation

Starts the execution of commands registered in OT, indicated by *ot*. Unlike `GsDrawOt()`, the processing is performed in the foreground; thus this function does not return until drawing is completed.

Mainly used for debugging.

See also

[GsDrawOt\(\)](#)

GsGetActiveBuffer

Get a buffer number during drawing.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
int GsGetActiveBuffer(void)
```

Explanation

Gets a double buffer index. Index values are either 0 or 1.

By entering indexes in the external variables, PSDBASEX[] and PSDBASEY[], it is possible to determine the two-dimensional address of the double buffer origin point (top left coordinates) in the frame buffer.

Return value

Index of a double buffer (0 for buffer 0, and 1 for buffer 1)

See also

[GsDefDispBuff\(\)](#), [GsSwapDispBuffer\(\)](#)

GsGetLs

Calculate a local screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	9/1/99

Syntax

```
void GsGetLs(  
  GsCOORDINATE2 *coord,    Pointer to local coordinates  
  MATRIX *m)               Pointer to matrix
```

Explanation

Calculates a local screen perspective transformation matrix from the GsCOORDINATE2 structure pointed to by the *coord* argument and stores the result in the MATRIX structure pointed to by the *m* argument.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When the next GsGetLs() function is called, calculation up to the node to which no changes have been made is omitted. This is controlled by a GsCOORDINATE2 member flag (*libgs* replaces the external variable PSDCNT in flags already calculated by GsCOORDINATE2).

If the contents of a superior node are changed, the effect on a subordinate node is handled by *libgs*, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

See also

[GsGetLw\(\)](#), [GsGetLws\(\)](#)

GsGetLw

Calculate a local world matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	9/1/99

Syntax

```
void GsGetLw(
  GsCOORDINATE2 *coord,    Pointer to local coordinate system
  MATRIX *m)               Pointer to matrix
```

Explanation

Calculates a local world perspective transformation matrix from the GsCOORDINATE2 structure pointed to by the *coord* argument and stores the result in the MATRIX structure pointed to by the *m* argument.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When the next GsGetLw() function is called, calculation up to the node to which no changes have been made is omitted. This is controlled by a GsCOORDINATE2 member flag (libgs replaces the external variable PSDCNT in flags already calculated by GsCOORDINATE2).

If the contents of a superior node are changed, the effect on a subordinate node is handled by libgs, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

See also

[GsGetLs\(\)](#), [GsGetLws\(\)](#)

GsGetLws

Calculate local world and local screen matrices.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsGetLws(
  GsCOORDINATE2 *coord2,    Pointer to local coordinates
  MATRIX *lw,               Pointer to matrix that stores the local world coordinates
  MATRIX *ls)               Pointer to matrix that stores the local screen coordinates
```

Explanation

GsGetLws() calculates local world and local screen coordinates. It is faster than calling GsGetLw() followed by calling GsGetLs(). When you use GsSetLightMatrix(), you pass it the *lw* matrix.

See also

[GsGetLw\(\)](#), [GsGetLs\(\)](#), [GsSetLightMatrix\(\)](#)

GsGetTimInfo

Find TIM format header.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsGetTimInfo(
  u_long *tim,           Pointer to TIM data
  GsIMAGE *im)          Pointer to an image Structure
```

Explanation

Fills in the GsIMAGE structure pointed to by the *im* parameter with the appropriate information obtained from the TIM data located at the address specified by the *tim* parameter.

GsGetVcount

Get the value of the vertical retrace counter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

long GsGetVcount(void)

Explanation

Gets the value of the vertical retrace counter.

Return value

Value of the vertical retrace counter.

See also

[GsClearVcount\(\)](#), [GsInitVcount\(\)](#)

GsGetWorkBase

Get address for storing current drawing commands.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

PACKET *GsGetWorkBase(void)

Explanation

Allocates and returns a pointer to a buffer used for generating a drawing primitive GPU packet.

Return value

Address to prepare the next drawing primitive packet.

GsInit3D

Initialize the graphics system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

void GsInit3D(void)

Explanation

Initializes the libgs three-dimensional graphics system. It must be called before using other 3D functions such as GsSetRefView2(), GsInitCoordinate2(), and GsSortObject3(). It does the following:

- Brings the screen point of origin to the center of the screen.
- Sets the light source to the LIGHT_NORMAL default.

GsInitCoordinate2

Initialize a local coordinate system

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsInitCoordinate2(
GsCOORDINATE2 *super,   Pointer to a superior coordinate system
GsCOORDINATE2 *base)    Pointer to a coordinate system (to be initialized)
```

Explanation

Initializes the coordinate system *base*. *base->coord* is set to an identity matrix (GsIDMATRIX). *super->sub* is set to *base*.

GsInitFixBg16, GsInitFixBg32

Initialize BG work area (high-speed)

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```

void GsInitFixBg16(
  GsBG *bg,           Pointer to GsBG
  u_long *work)       Pointer to work area (primitive area)

```

```

void GsInitFixBg32(
  GsBG *bg,           Pointer to GsBG
  u_long *work)       Pointer to work area (primitive area)

```

Explanation

These functions initialize the work area used by the functions GsSortFixBg16() and GsSortFixBg32(), respectively. The size of the array differs with the screen mode as follows:

- size (in long units)=(((ScreenW/CellW+1) x (ScreenH/CellH+1+1) x 6+4) x 2+2)
- ScreenH: screen height in pixels (240/480)
- ScreenW: screen width in pixels (256/320/384/512/640)
- CellH: cell height (in pixels)
- CellW: cell width (in pixels)

Executing GsInitFixBg16()/GsInitFixBg32() once is sufficient; you need not execute it for every frame.

See also

[GsSortFixBg16\(\)](#)

GsInitGraph

Initialize the graphics system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsInitGraph(
  u_short x_res,      Horizontal resolution (256/320/384/512/640)
  u_short y_res,      Vertical resolution (240/480 NTSC or 256/512 PAL)
  u_short int1,        see Explanation
  u_short dither,      Dithering processing flag. 0: OFF, 1: ON
  u_short vram)        VRAM mode. 0: 16-bit, 1: 24-bit
```

Explanation

Resets libgpu and initializes the libgs graphic system. libgpu settings are maintained by the global variables GsDISPENV and GsDRAWENV. The programmer can verify and/or modify libgpu by referencing the settings.

The bits of the *int1* argument are as follows:

- Interlace display flag (bit 0)
0: Non-interlace GsNONINTER
1: Interlace GSINTER
- Double buffer offset mode (bit 2)
0: GTE offset GsOFSGTE
1: GPU offset GsOFSGPU
- GPU Initialize Parameter (bits 4-5)
0: ResetGraph(0) GsRESET0
3: ResetGraph(3) GsRESET3

Vertical 480 line non-interlace mode is effective only when a VGA monitor is connected. In 240-line mode, the top and bottom 8 lines are almost invisible on home-use TV monitors. For PAL mode, the display position should be shifted down by 24 lines.

The double buffer offset mode is either GTE or GPU offset; when it is GPU, the packet does not include the offset value and therefore be handled easily.

For 24-bit mode, only the memory image display is available and polygon drawing cannot be done.

Since initialization of the graphic system involves initialization of GsIDMATRIX and GsIDMATRIX2 as well, GsInitGraph() must be called prior to all other libgs functions for correct operation.

See also

[GsInitGraph2\(\)](#)

GsInitGraph2

Initialize the graphics system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

void GsInitGraph2(

u_short *x_res*,

u_short *y_res*,

u_short *int1*,

u_short *dither*,

u_short *vram*)

Horizontal resolution (256/320/384/512/640)

Vertical resolution (240/480)

Interlace display flag (bit 0) 0: Non-interlace, 1: Interlace

Double buffer offset mode (bit 2) 0: GTE offset, 1: GPU offset

Dithering. 0: OFF, 1: ON

VRAM mode. 0: 16-bit, 1: 24-bit

Explanation

GsInitGraph2() is different from GsInitGraph() in that the GPU is not initialized COLD. This function is useful for changing libgs resolution without affecting screen synchronization.

Always use GsInitGraph() for the first initialization.

See also

[GsInitGraph\(\)](#)

GsInitVcount

Initialize vertical retrace counter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsInitVcount(void)
```

Explanation

Initializes the vertical retrace counter, and starts it.

See also

[GsClearVcount\(\)](#), [GsGetVcount\(\)](#)

GsLinkObject3

Link an object with PMD data; for GsSortObject3().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsLinkObject3(
  u_long *pmd,           Pointer to the PMD data to be linked
  GsDOBJ3 *obj_base)    Pointer to the object structure to be linked
```

Explanation

Links all objects contained in PMD data to a GsDOBJ3 object structure.

Note: Unlike GsLinkObject4(), it is not possible to select and link a single object in the PMD data.

See also

[GsSortObject3\(\)](#), [GsLinkObject4\(\)](#)

GsLinkObject4

Link an object to TMD data; for GsSortObject4().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsLinkObject4(
  u_long tmd,           Starting address of the TMD data to be linked
  GsDOBJ2 *obj_base,    Array of the object structure to be linked
  int n)                Index of the object to be linked
```

Explanation

Links the n -th object of TMD-format three-dimensional data to a GSDOBJ2 object structure.

An object linked using this function uses GsSortObject4() to create a packet.

See also

[GsSortObject4\(\)](#)

GsLinkObject5

Link an object to TMD data; or GsSortObject5().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsLinkObject5(
u_long tmd,           Starting address of the TMD data to be linked
GsDOBJ5 *obj_base,    Array of the object structure to be linked
int n)                Index of the object to be linked
```

Explanation

Links the n -th object of TMD-format three-dimensional data to a GSDOBJ5 object structure.

An object linked using this function uses GsSortObject5() to create a packet.

See also

[GsSortObject5\(\)](#)

GsMapModelingData

Map TMD data to real addresses.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsMapModelingData(
u_long *p)           Pointer to starting address of TMD data
```

Explanation

TMD data includes several fields containing addresses of data. During the preparation of TMD data, the load addresses of the data are not yet known; therefore, the address fields are stored as offsets from the start of the data. GsMapModelingData() changes these offsets into actual addresses after the TMD data has been loaded into memory, so that the TMD data may be used.

A flag is set in the TMD data to indicate when offset addresses have been converted into real addresses. Therefore, no side effect occurs even if GsMapModelingData() is called again.

GsMulCoord0

MATRIX multiplication.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

**void GsMulCoord2(
MATRIX **m1*, MATRIX **m2*)** Pointers to starting addresses of TMD data

Explanation

Multiplies MATRIX *m2* by the translation matrix. The results are stored in *m3*.

$m3 = m1 \times m2$

See also

[GsMulCoord2\(\)](#), [GsMulCoord3\(\)](#)

GsMulCoord2

MATRIX multiplication.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

**void GsMulCoord2(
MATRIX **m1*, MATRIX **m2*)** Pointers to matrices

Explanation

GsMulCoord2 multiplies the MATRIX *m2* by the translation matrix *m1* and stores the result in *m2*.

$m2 = m1 \times m2$

See also

[GsMulCoord0\(\)](#), [GsMulCoord3\(\)](#)

GsMulCoord3

MATRIX multiplication.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsMulCoord3(
  MATRIX *m1, MATRIX *m2)  Pointers to matrices
```

Explanation

GsMulCoord3 multiplies the MATRIX *m2* by the translation matrix *m1* and stores the result in *m2*.

$m1 = m1 \times m2$

See also

[GsMulCoord0\(\)](#), [GsMulCoord2\(\)](#)

GsPresetObject

Create a preset packet for a GsDOBJ5-type object.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
u_long *GsPresetObject(
  GsDOBJ5 *objp,           Pointer to the object to be preset
  u_long *addr)             Pointer to starting address of the area in which the preset packet is to be
                             prepared.
```

Explanation

Presetting refers to the advance preparation of polygons of all objects as packets. The areas that need not be rewritten (e.g., U and V of texture) for each frame are not rewritten, thus ensuring high speed.

The return value points to the address next to the last preset address, so when presetting the next object, preserve the return value and pass it as an argument of the next GsPresetObject(). The return value indicates how large an area must be allocated for the preset area.

A GsDOBJ5 type object pointer is exclusively used for presetting.

Return value

Pointer that indicates the next to last preset address.

See also

[GsPrst...\(\)](#)

GsPrst...

Low-level functions for GsSortObject5J().

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	12/14/98

Syntax

PACKET *GsPrst...(
TMD_P_... *op, Pointer to starting address of TMD data primitives
VERT *vp, Pointer to starting address of TMD data vertices TMD
VERT *np, Pointer to starting address of TMD data normals
PACKET *pk, Pointer to top address of GPU packet buffer
int n, Number of primitives
int shift, OT shift bit
GsOT *ot, Pointer to GsOT
u_long *scratch) Pointer to starting address of unused scratch pad

PACKET *GsPrstN...(
TMD_P_... *op, Pointer to starting address of TMD data primitives
VERT *vp, Pointer to starting address of TMD data vertices TMD
PACKET *pk, Pointer to top address of GPU packet buffer
int n, Number of primitives
int shift, OT shift bit
GsOT *ot, Pointer to GsOT
u_long *scratch) Pointer to starting address of unused scratch pad

Explanation

These are low-level functions for GsSortObject5J().

To use these functions, they must be registered in GsFCALL5 as low-level functions.

These functions perform coordinate and perspective transformation, backface clipping, and light source calculation for *n* primitives, create the GPU packet in the buffer, and link it into the OT. There must be two preset packets in the buffer per polygon.

For function types which do not operate on normals within the data (e.g. GsPrstN...), light source calculations are not performed, so fewer parameters are passed compared to those function types which operate on normals (e.g. GsPrst...),

Low-level functions in libgs that are supported are shown below.

Table 9-7: GsPrst...() [have normals]

Low-level function name	First arg (op) type	Description
GsPrstF3L	TMD_P_F3	Flat triangle (light source calculation)
GsPrstF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsPrstF3NL	TMD_P_F3	Flat triangle
GsPrstF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsPrstF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsPrstF4NL	TMD_P_F4	Flag quadrilateral
GsPrstG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsPrstG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsPrstG3NL	TMD_P_G3	Gouraud triangle
GsPrstG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsPrstG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsPrstG4NL	TMD_P_G4	Gouraud quadrilateral

Low-level function name	First arg (op) type	Description
GsPrstTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsPrstTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsPrstTF3NL	TMD_P_TF3	Textured flat triangle
GsPrstTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsPrstTF4LFG	TMD_P_TF4	Flat quadrilateral (light source calculation +FOG)
GsPrstTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsPrstTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsPrstTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsPrstTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsPrstTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsPrstTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsPrstTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsPrstF3GL	TMD_P_F3G	Flat gradation triangle (light source calculation)
GsPrstF3GLFG	TMD_P_F3G	Flat gradation triangle (light source calculation +FOG)
GsPrstF3GNL	TMD_P_F3G	Flat gradation triangle
GsPrstG3GL	TMD_P_G3G	Gouraud gradation triangle (light source calculation)
GsPrstG3GLFG	TMD_P_G3G	Gouraud gradation triangle (light source calculation +FOG)
GsPrstG3GNL	TMD_P_G3G	Gouraud gradation triangle

Table 9-8: GsPrstN...() [no normals]

Low-level function name	First arg (op) type	Description
GsPrstNF3	TMD_P_NF3	Flat triangle
GsPrstNF4	TMD_P_NF4	Flat quadrilateral
GsPrstNG3	TMD_P_NG3	Gouraud triangle
GsPrstNG4	TMD_P_NG4	Gouraud quadrilateral
GsPrstTNF3	TMD_P_TNF3	Textured flat triangle
GsPrstTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsPrstTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsPrstTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral

Return value

Starting address of unused packet area.

With gradation, each vertex of the TMD polygon has a different RGB value.

For high speed operation, libgte contains tuned assembly-level low-level functions.

See also

[GsPresetObject\(\)](#), [GsSortObject5J\(\)](#)

GsScaleScreen

Scale the screen coordinate system.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

```
void GsScaleScreen(  
SVECTOR *scale)
```

Pointer to the scale factor (12 bit fixed radix point format). Always set the factor in relation to the original screen coordinate systems set on GsSetView2() and GsSetRefView2(). When ONE (4096) is inserted into *scale->vx*, *vy* or *vz*, it returns to the original scale.

Explanation

Scales the screen coordinate system against the world coordinates.

World coordinates are 32-bit values and screen coordinates are 16-bit values. This difference brings about problems such as FarClip being close.

To solve these problems, this function scales the screen coordinates to cover a larger area than world.

For example, when specifying ONE/2 to *vx*, *vy* or *vz*, the screen coordinate system is expanded to the equivalent of 17 bits. However, since the precision is 16 bits, the lower 1 bit is invalid.

Note: Make sure that the screen coordinate system which has a different scale is not registered to the OT with the same scale. For example, before registering an object calculated with the normal scaling screen coordinate system to the OT which has already registered an object with a 1/2 screen coordinate system scale, it is necessary to shift the excess 1 bit.

When the scaling matrix set by this function to the external variable GsWSMATRIX, and the screen coordinates set by GsSetView2() and GsSetRefView2() to the external variable GsWSMATRIX_ORG are defined, the WSMATRIX is held.

GsSetAmbient

Set color and brightness of ambient lighting.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSetAmbient(
u_long r, g, b)           Ambient color RGB values (0-4095)
```

Explanation

Sets the color and brightness of the ambient lighting in the 3D world. Values for red, green, and blue are set independently. A value of 4096 corresponds to normal ambient brightness, 0 to minimum brightness. Values greater than 4096 strengthen that color. For example, 1/1 is 4096 and 1/8 is 4096/8.

GsSetClip

Set drawing clipping area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

**void GsSetClip(
RECT *clip)** Beginning address of a RECT structure for setting a clipping area

Explanation

Sets clipping for drawing. This function is different from GsSetDrawBuffClip() in that its argument can be used to specify a clip area. Note that this clipping value is a relative one within the double buffer, and thus the clip position doesn't change if the double buffer is swapped.

Clipping is done by libgpu.

See also

[GsSetDrawBuffClip\(\)](#), [GsSetClip2\(\)](#)

GsSetClip2

Set a drawing clipping area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

DRAWENV *GsSetClip2(
RECT *clip) Beginning address of a RECT structure for setting a clipping area

Explanation

Sets the clipping rectangle for drawing to the rectangle specified by *clip*. This function is different from GsSetClip() in that the DRAWENV and DISPENV structures are not updated. The return value of GsSetClip2() is a pointer to a DRAWENV structure that can be used if necessary to set the system DRAWENV structure using PutDrawEnv(). The global DRAWENV must have been previously specified in order for the information in this structure to be valid.

Note: This clipping rectangle is relative to whichever is the current buffer, even if double-buffering is used.

Return value

A pointer to an updated DRAWENV structure (which can be used to update the system DRAWENV structure if desired).

Clipping is done by libgpu.

See also

PutDrawEnv(), [GsSetClip\(\)](#)

GsSetClip2D

Set two-dimensional clipping.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

GsSetClip2D(
RECT **rectp*) Pointer to the area to be clipped

Explanation

Sets the area given by *rectp* as the area to be clipped.

When swapping double buffers, clipping occurs in the same relative position in both buffers.

GsSetDrawBuffClip() must be called in order to validate this setting immediately afterwards. If it is not called, the setting is valid from the next frame.

See also

[GsSetDrawBuffClip\(\)](#)

GsSetDrawBuffClip

Set drawing clipping area.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSetDrawBuffClip(void)
```

Explanation

Sets clipping for drawing. The clipping value set by GsSetClip2D() is set in libgs.

This value is a relative one within the double buffers, so the clipping position does not change when buffers are swapped.

This function does not execute correctly if GPU drawing is in progress. Use ResetGraph(1) to terminate any current drawing process or DrawSync() to wait until the process is completed.

See also

[GsSetClip2D\(\)](#), [ResetGraph\(\)](#), [DrawSync\(\)](#)

GsSetDrawBuffOffset

Set the drawing offset.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

void GsSetDrawBuffOffset(void)

Explanation

Sets the drawing offset stored in the global variable POSITION. This offset is relative within the double buffer, so it is preserved if the buffers are swapped.

It sets the libgte or libgpu offset, depending on the value of the third argument of GsInitGraph(), either GsOFSGPU or GsOFSGTE.

This function does not execute correctly if GPU drawing is in progress. Use ResetGraph(1) to terminate any current drawing process or DrawSync() to wait until the process is completed.

See also

ResetGraph(), DrawSync(), [GsSetOffset\(\)](#)

GsSetFlatLight

Set a parallel light source.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

void GsSetFlatLight(

u_int *id*, Light source number (0, 1, 2)
GsF_LIGHT **lt*) Pointer to light source data

Explanation

Sets the values for one of up to three parallel light sources. Light source data is specified in the GsF_LIGHT structure.

GsSetFogParam

Set the fog parameter.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

**void GsSetFogParam(
[GsFOGPARAM](#) *fogparam)** Pointer to a fog parameter structure

Explanation

Sets the fog parameter. Fog is valid only in lighting modes 1 and 3. (However, lighting mode 3 is currently not supported.)

GsSetLightMatrix

Set a light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSetLightMatrix(
MATRIX *mp)           Pointer to matrix
```

Explanation

The three light source vectors and the local screen light matrix *mp* are multiplied and placed in the GTE. When using libgte to do light source calculations, call GsSetLightMatrix() first.

Depending on the type of model data, some of the GsSortObject...() routines do light source calculations (there are no preset light calculations). In this case, also, you must use GsSetLightMatrix() to set a light matrix in advance.

Generally, *mp* is a local-world matrix.

See also

[GsSetLightMatrix2\(\)](#)

GsSetLightMatrix2

Set a light matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsSetLightMatrix2(
MATRIX *mp)           Pointer to matrix
```

Explanation

The three light source vectors and the local screen light matrix *mp* are multiplied and placed in the GTE. When using libgte to do light source calculations, call GsSetLightMatrix() first.

Depending on the type of model data, some of the GsSortObject...() routines do light source calculations (there are no preset light calculations). In this case, also, you must use GsSetLightMatrix2() to set a light matrix in advance.

Generally, *mp* is a local-world matrix.

The difference between GsSetLightMatrix() and this function is whether the GTE rotation matrix and the parameter *mp* are destroyed or not. GsSetLightMatrix2() destroys these values, however, GsSetLightMatrix2() is faster than GsSetLightMatrix().

You must call GsSetLightMatrix() before GsSetLsMatrix().

See also

[GsSetLightMatrix\(\)](#), [GsSetLsMatrix\(\)](#)

GsSetLightMode

Set light source mode.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSetLightMode(
int mode)
```

Light source mode value:

0: Normal lighting

1: Normal lighting with fog ON

2: Material lighting (not currently supported)

3: Material lighting with fog ON (not currently supported)

Explanation

Sets the default light source mode. The method of light source calculation can be also set using status bits for each object. The setting of the status bit overrides the default setting.

GsSetLsMatrix

Set a local screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

**void GsSetLsMatrix(
MATRIX *mp)** Pointer to local screen matrix to be set

Explanation

Sets a GTE local screen matrix. When you use this function for libgte perspective transform processing, you must first set a local screen matrix in libgte.

For GsSortObject...() calls to perform perspective transformations and use them in libgte, you must first call this function.

See also

[GsSetLightMatrix2\(\)](#)

GsSetOffset

Set an offset.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsSetOffset(
long offx,          Drawing offset X
long offy)          Drawing offset Y
```

Explanation

Specifies a drawing offset. This function is different from GsSetDrawBuffOffset() in that it sets an offset provided as an argument while GsSetDrawBuffOffset() sets a value for the global variable, POSITION. The offset to be provided as an argument is a relative offset inside the double buffer. In other words, the double buffer base offset is added to the offset provided by the argument.

Using the GsOFSGPU or GsOFSGTE macro for the third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set.

This function does not execute correctly if GPU drawing is in progress. Use ResetGraph(1) to terminate any current drawing process of DrawSync() to wait until the process is completed.

See also

[GsSetDrawBuffOffset\(\)](#)

GsSetOrign

Set offset that is valid if the screen is switched.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

```
void GsSetOrign(
long x ,           Drawing offset X
long y)           Drawing offset Y
```

Explanation

Specifies a drawing offset. This offset value is valid until the GsSetOrign() is called again, unlike GsSetOffset(), where the offset value is temporary and becomes invalid when GsSwapDispBuff() and GsSetDrawBuffOffset() are called.

The *x*, *y* offset provided is relative inside the double buffer; that is, the double buffer base offset is added to the offset provided.

Note: The third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set (either GsOFSGPU or GsOFSGTE)

This function does not execute correctly when GPU drawing is in progress.

See also

[GsInitGraph\(\)](#), [GsSetClip2D\(\)](#)

GsSetProjection

Set the projection plane position.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

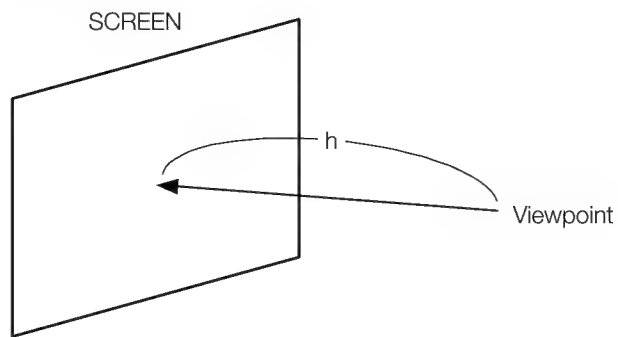
Syntax

```
void GsSetProjection(
u_long h)           Distance (projection) between the viewpoint and projection plane
                    Default: 1000
```

Explanation

Sets the distance between the projection plane and the viewpoint. This results in a change in field of view.

Figure 9-1: Projection



The size of the projection plane is specified by (*x_res*, *y_res*) in `GsInitGraph()`. The size of the projection plane is constant with respect to the resolution, so the drawing angle is reduced as projection is increased, and the drawing angle is increased as projection is decreased.

See also

[GsInitGraph\(\)](#)

GsSetRefView2

Set world-to-screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
int GsSetRefView2(
  GsRVIEW2 *pv)      Pointer to view information
```

Explanation

Calculates GsWSMATRIX using viewpoint information in *pv*. GsWSMATRIX doesn't change unless the viewpoint is moved, so this function should be called every frame only if the viewpoint is moved, in order for changes to be updated.

It should also be called every frame if the GsRVIEW2 member *super* is set to anything other than WORLD, because even if the other parameters are not changed, if the parameters of the superior coordinate system are changed, the viewpoint will have moved.

Return value

0 on success; 2 on failure.

See also

[GsSetRefView2L\(\)](#)

GsSetRefView2L

Set viewpoint (high-precision version).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.5	12/14/98

Syntax

```
int GsSetRefView2L(
  GsRVIEW2 *pv)      Pointer to viewpoint location information (view/reference point type)
```

Explanation

Calculates GsWSMATRIX using the viewpoint information in *pv*. GsWSMATRIX doesn't change unless the viewpoint is moved, so this function should be called every frame only if the viewpoint is moved, in order for changes to be updated.

It should also be called every frame if the GsRVIEW2 member *super* is set to anything other than WORLD, because even if the other parameters are not changed, if the parameters of the superior coordinate system are changed, the viewpoint will have moved.

Compared to GsSetRefView2(), GsSetRefView2L() has higher precision: viewpoint wobbling caused by insufficient precision is improved. However, its execution time is doubled.

Return value

0 for successful viewpoint set, 1 for error.

See also

[GsSetRefView2\(\)](#)

GsSetView2

Set viewpoint.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
int GsSetView2(
  GsVIEW2 *pv)      Pointer to viewpoint position data (matrix form)
```

Explanation

Sets GsWSMATRIX directly.

If you use GsSetRefView2() to determine the WS matrix from the viewpoint and the focal point, insufficient precision may cause errors when you move the viewpoint; it is more effective to use GsSetView2().

If the GsVIEW2 *super* member is anything besides WORLD, you must call this function in each frame in which the parent coordinate system parameters are changed.

If GsIDMATRIX2 is used as the base matrix, then the aspect ratio of the screen is adjusted automatically.

Return value

0 if successful; 1 if unsuccessful.

See also

[GsSetRefView2\(\)](#)

GsSetWorkBase

Set address for storing drawing commands.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSetWorkBase(
PACKET *base_addr)    Pointer to an address storing drawing commands
```

Explanation

Sets the memory address for storing drawing primitives generated by functions like GsSortObject...(), GsSortSprite(), and GsSortBg().

Primitives must be stored at the starting address of a packet area reserved by the user at the beginning of processing for each frame.

See also

[GsSortSprite\(\)](#), [GsSortBg\(\)](#)

GsSortBg, GsSortFastBg

Register BG in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```

void GsSortBg(
GsBG *bg,           Pointer to BG
GsOT *otp,          Pointer to OT
u_short pri)        Position in OT

```

```

void GsSortFastBg(
GsBG *bg,           Pointer to BG
GsOT *otp,          Pointer to OT
u_short pri)        Position in OT

```

Explanation

Assigns BG indicated by *bg* to the ordering table indicated by *otp*. *pri* refers to the priority of the Sprite in the ordering table. The highest priority is zero, with the lowest priority depending on the size of the ordering table. Values beyond the ordering table size are clipped to the available maximum value.

Turning off extension and rotation functions in the bg attributes gives higher-speed processing.

In GsSortFastBg(), not using enlargement, rotation, and reduction functions results in higher-speed processing. The Sprite structure members values *mx*, *my*, *scalex*, *scaley*, and *rotate* are ignored.

See also

[GsSortFixBg16\(\)](#)

GsSortBoxFill

Register a rectangle in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSortBoxFill(
  GsBOXF *bp,      Pointer to GsBOXF
  GsOT *ot,         Pointer to OT
  u_short pri)      Position in OT
```

Explanation

Assigns a rectangle indicated by *bp* to the ordering table indicated by *ot*.

GsSortClear

Register a screen clear command in the OT.

Library	Header File	Introduced	Documentation Date
libgs.lib	libgs.h	2.x	12/14/98

Syntax

```
void GsSortObject(
u_char r, u_char g, u_char b,   Background color RGB values
GsOT *otp)                     Pointer to OT
```

Explanation

Sets a screen clear command at the start of the OT indicated by *otp*. Should be called after GsSwapDispBuff(). **Note:** Actual clearing isn't executed until GsDrawOt() is used to start drawing.

See also

[GsSwapDispBuff\(\)](#), [GsDrawOt\(\)](#)

GsSortFixBg16, GsSortFixBg32

Register BG in the OT (high-speed)

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

void GsSortFixBg16(

[GsBG](#) *bg, Pointer to GsBG
u_long *work, Pointer to work area (primitive area)
[GsOT](#) *otp, Pointer to OT
u_short pri) Position in OT

void GsSortFixBg32(

[GsBG](#) *bg, Pointer to GsBG
u_long *work, Pointer to work area (primitive area)
[GsOT](#) *otp, Pointer to OT
u_short pri) Position in OT

Explanation

These functions perform high-speed BG registration. They are less CPU-intensive than GsSortFastBg(), with the following restrictions.

- BG rotation/enlargement/reduction is not possible
- Fixed cell size: 16 for GsSortFixBg16(), 32 for GsSortFixBg32()
- Texture pattern color mode is only 4-bit/8-bit
- Map size is optional
- Scroll is possible (in 1-pixel units)
- Only full-screen

These functions use the work area to store drawing primitives. The work area uses an unsigned long array; this must be initialized beforehand by GsInitFixBg16() or GsInitFixBg32(). These functions do not use the packet area (an area set by GsSetWorkBase()).

See also

[GsSortBg\(\)](#), [GsSetWorkBase\(\)](#)

GsSortGLine, GsSortLine

Register a straight line in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```

void GsSortGLine(
    GsLINE *lp,           Pointer to GsLINE/GsGLINE
    GsOT *ot,             Pointer to OT
    u_short pri)          Position in OT
    
```

```

void GsSortLine(
    GsLINE *lp,           Pointer to GsLINE/GsGLINE
    GsOT *ot,             Pointer to OT
    u_short pri)          Position in OT
    
```

Explanation

Assigns the straight line indicated by *lp* to the ordering table indicated by *ot*.
 GsSortLine() registers single-color straight lines in OT, and GsSortGLine() graded straight lines in OT.

GsSortObject3

Register an object to the ordering table (for use with GsDOBJ3).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

void GsSortObject3(

GsDOBJ3 **objp*,

GsOT **otp*,

int *shift*)

Pointer to an object

Pointer to OT

Specifies how many bits the Z value must be shifted to the right when assigning an object to the OT

Explanation

Performs perspective transformation and light source calculation for a three-dimensional object handled by GsDOBJ3, and creates a drawing command within the PMD format packet memory. Performs Z-sort of the drawing commands generated immediately afterwards and assigns them to the OT indicated by *otp*.

The accuracy of Z may be adjusted with the value of *shift*. The maximum size of the ordering table (resolution) is 14 bits, but if this value is set to 12 bits, for example, the shift value must be set at 2 (=14-12), so that it will not be larger than the ordering table area.

GsSortObject4

Register an object to the ordering table (for use with GsDOBJ2).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsSortObject4(
  GsDOBJ2 *objp,      Pointer to an object
  GsOT *otp,           Pointer to OT
  int shift,           Specifies how many bits the Z value must be shifted to the right when
                      assigning an object to the OT
  u_long *scratch)     Pointer to address of scratch pad
```

Explanation

Performs perspective transformation and light source calculation for a three-dimensional object handled by GsDOBJ2, and creates a drawing command within the packet area specified by GsSetWorkBase(). Performs Z-sort of the drawing commands generated immediately afterwards and assigns them to the OT indicated by *otp*.

The accuracy of Z may be adjusted with the value of *shift*. The maximum size of the ordering table (resolution) is 14 bits. If this value is set to 12 bits, for example, the shift value must be set at 2 (=14-12), so that it will not be larger than the ordering table area.

scratch is the specified scratchpad address used as work when automatic division is being performed. The scratchpad runs for 256 words from 0x1f800000 in cache memory.

To use the GsOBJ2 member attribute to enable division, perform an OR operation on the macros GsDIV1 through GsDIV5 (defined in libgs.h). For GsDIV1, a single polygon is divided into four segments of 2 x 2. For GsDIV5, a single polygon is divided into 1,024 segments of 32 x 32.

For a triangle, the scratch area usage is $96 + 88 \times N$, where N is the number of the macro used (GsDIV1 = 1, GsDIV2 = 5, etc.) For a quadrilateral, the scratch area used is $120 + 140 \times N$.

See also

[GsSortObject4J\(\)](#)

GsSortObject4J

Register an object to ordering table (jump table version).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

```
void GsSortObject4J(  
  GsDOBJ2 *objp,           Pointer to an object  
  GsOT *otp,               Pointer to OT  
  int shift,               Specifies how many bits the Z value must be shifted to the right when  
                           assigning an object to the OT  
  u_long *scratch)         Pointer to the address of the scratch pad
```

Explanation

Same functionality as GsSortObject4(), when all the low-level functions have been registered. Allows programmer to increase code efficiency by not calling unnecessary low-level functions (up to 40 kbytes can be saved.)

To do this, test which low-level routines are being called by prepending 'dmy' to the function names in GsFCALL4, the reference function table. The names of all functions called are printed out; for those functions, delete 'dmy', and only those functions used will be linked in.

See also

[GsSortObject4\(\)](#), [GsSortObject5J\(\)](#)

GsSortObject5

Register an object to the ordering table (for use with GsDOBJ5).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

```
void GsSortObject5(
  GsDOBJ5 *objp,      Pointer to an object
  GsOT *otp,           Pointer to OT
  int shift,           Specifies how many bits the Z value must be shifted to the right when
                      assigning an object to the OT
  u_long *scratch)     Pointer to the address of the scratch pad
```

Explanation

Performs transparency transformation and light source calculation for a three-dimensional object to be handled by GsDOBJ5. It creates in the preset packet area drawing commands that do not divide, and in the packet area specified by GsSetWorkBase() those drawing commands that do divide. Performs Z-sort of the drawing commands generated immediately afterwards and assigns them to the OT indicated by *otp*.

The accuracy of Z may be adjusted using the *shift* value. The maximum size of the ordering table (resolution) is 14 bits. If this value is set to 12 bits, for example, the shift value must be set at 2 (=14-12), so that it will not be larger than the ordering table area.

scratch is the specified scratchpad address used as work when automatic division is being performed. (The scratchpad is 256 words starting from 0x1f800000 in cache memory.) To use GsDOBJ5.*attribute* to enable division, perform an OR operation on the macros GsDIV1-GsDIV5 of libgs.h. For GsDIV1, a single polygon is divided into four segments of 2 x 2. For GsDIV5, a single polygon is divided into 1, 024 segments of 32 x 32.

For a triangle, the scratch area usage is $96 + 88 \times N$, where N is the number of the macro used (GsDIV1 = 1, GsDIV2 = 5, etc.) For a quadrilateral, the scratch area used is $120 + 140 \times N$.

See also

[GsSortObject5J\(\)](#)

GsSortObject5J

Register an object to the ordering table (jump table version).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

```
void GsSortObject5J(  
  GsDOBJ5 *objp,           Pointer to an object  
  GsOT *otp,               Pointer to OT  
  int shift,               Specifies how many bits the Z value must be shifted to the right when  
                           assigning an object to the OT  
  u_long *scratch)         Pointer to the address of the scratch pad
```

Explanation

Same functionality as GsSortObject5(), when all the low-level functions have been registered. Allows programmer to increase code efficiency by not calling unnecessary low-level functions (up to 40 kbytes can be saved.)

To do this, test which low-level routines are being called by prepending 'dmy' to the function names in GsFCALL5, the reference function table. The names of all functions called are printed out; for those functions, delete 'dmy', and only those functions used will be linked in.

See also

[GsSortObject5\(\)](#), [GsSortObject4J\(\)](#)

GsSortOt

Insert an OT into another OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.3	12/14/98

Syntax

```
GsOT *GsSortOt(
GsOT *ot_src,      Pointer to source OT
GsOT *ot_dest)     Pointer to destination OT
```

Explanation

Inserts the OT given by *ot_src* into the OTZ location within *ot_dest*.

The OTZ value used at this time is calculated as follows:

OTZ = *ot_src*->point - *ot_dest* ->offset

Return value

Pointer to the integrated OT.

See also

[GsClearOt\(\)](#), [GsDrawOt\(\)](#), [GsCutOt\(\)](#)

GsSortPoly

Register a polygon drawing primitive in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

```
void GsSortPoly(
void *prim,           Pointer to drawing primitive
GsOT *ot,             Pointer to OT
u_short pri)          Location in OT
```

Explanation

Assigns the drawing primitive *prim* to the ordering table *ot*. Corresponds to libgpu drawing primitives (POLY,...).

libgs requires no double buffering, since the contents of the primitive structure are copied in the packet generation area. Drawing coordinate values match the drawing coordinate system handled by libgs.

GsSortSprite, GsSortFastSprite, GsSortFlipSprite

Register a Sprite in the OT.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

void GsSortSprite(

GsSPRITE **sp*, Pointer to a Sprite
GsOT **otp*, Pointer to OT
u_short *pri*) Position in OT

void GsSortFastSprite(

GsSPRITE **sp*, Pointer to a Sprite
GsOT **otp*, Pointer to OT
u_short *pri*) Position in OT

void GsSortFlipSprite(

GsSPRITE **sp*, Pointer to a Sprite
GsOT **otp*, Pointer to OT
u_short *pri*) Position in OT

Explanation

Assigns the sprite *sp* to the ordering table *otp*.

pri refers to the priority of the Sprite in the ordering table. The highest priority value is zero, with the lowest value depending on the size of the ordering table. Values beyond the size of the ordering table are clipped to the maximum ordering table value.

GsSortFastSprite() provides high-speed processing, though enlargement, rotation, and reduction cannot be used. The Sprite structure members *nx*, *my*, *scalex*, *scaley*, and *rotate* are ignored.

GsSortFlipSprite() does not use the enlargement / rotation / reduction functions, and only supports flipping.

Performing enlargement or rotation destroys the GTE rotation matrix.

GsSwapDispBuffer

Swaps double buffers.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	2.x	12/14/98

Syntax

void GsSwapDispBuffer(void)

Explanation

Exchanges the display buffer with the drawing buffer according to data set by GsDefDispBuff(). Normally, swapping is done immediately after beginning vertical blanking. This function

- Sets display starting address
- Cancels blanking
- Sets double buffer index
- Switches two-dimensional clipping
- Sets libgte or libgpu offset
- Sets libgs offset

Note: Using the GsOFSGPU or GsOFSGTE macro for the third argument of GsInitGraph() determines whether the libgte or libgpu offset should be set.

This function does not execute correctly when GPU drawing is in progress, so it is necessary to call this function after terminating drawing using ResetGraph (1).

See also

[GsDefDispBuff\(\)](#)

GsTMDdiv...

Low-level functions for GsSortObject4J() (performs fixed division).

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	4.1	12/14/98

Syntax

PACKET *GsTMDdiv...(
TMD_P_... *op, Pointer to starting address of TMD data primitives
VERT *vp, Pointer to starting address of TMD data vertices TMD
VERT *np, Pointer to starting address of TMD data normals
PACKET *pk, Pointer to top address of GPU packet buffer
int n, Number of primitives
int shift, OT shift bit
GsOT *ot, Pointer to GsOT
DIVPOLYGON... *divp) Pointer to DIVPOLYGON3 or DIVPOLYGON4

PACKET *GsTMDdivN...(
TMD_P_... *op Pointer to starting address of TMD data primitives
VERT *vp, Pointer to starting address of TMD data vertices TMD
PACKET *pk, Pointer to top address of GPU packet buffer
int n, Number of primitives
int shift, OT shift bit
GsOT *ot, Pointer to GsOT
DIVPOLYGON... *divp) Pointer to DIVPOLYGON3 or DIVPOLYGON4

Explanation

These are low-level functions for GsSortObject4J(). To be used, they must be registered in GsFCALL4 as low-level functions.

These functions perform fixed division based on the number of divisions (GsNDIV).

These functions perform polygon division (based on the value of GsNDIV), coordinate and perspective transformation, backface clipping, and light source calculation for *n* primitives, complete the GPU packet in the buffer, and link it into the OT.

ndiv=1: 2x2 division

ndiv=2: 4x4 division

ndiv=3: 8x8 division

ndiv=4: 16x16 division

ndiv=5: 32x32 division

For function types which do not operate on normals within the data (e.g. GsTMDdivN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsTMDdiv...),

Low-level functions in libgs that support fixed division are shown below.

Table 9-9: GsTMDdiv...() [have normals]

Low-level function name	First arg (op) type	Description
GsTMDdivF3L	TMD_P_F3	Flat triangle (light source calculation)
GsTMDdivF3LB	TMD_P_F3	Flat triangle (light source calculation + 2face)
GsTMDdivF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsTMDdivF3LFGB	TMD_P_F3	Flat triangle (light source calculation +FOG + 2face)
GsTMDdivF3NL	TMD_P_F3	Flat triangle
GsTMDdivF3NLB	TMD_P_F3	Flat triangle (2face)
GsTMDdivF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsTMDdivF4LB	TMD_P_F4	Flat quadrilateral (light source calculation + 2face)
GsTMDdivF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsTMDdivF4LFGB	TMD_P_F4	Flat quadrilateral (light source calculation +FOG + 2face)
GsTMDdivF4NL	TMD_P_F4	Flag quadrilateral
GsTMDdivF4NLB	TMD_P_F4	Flag quadrilateral (2face)
GsTMDdivG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsTMDdivG3LB	TMD_P_G3	Gouraud triangle (light source calculation + 2face)
GsTMDdivG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsTMDdivG3LFGB	TMD_P_G3	Gouraud triangle (light source calculation +FOG + 2face)
GsTMDdivG3NL	TMD_P_G3	Gouraud triangle
GsTMDdivG3NLB	TMD_P_G3	Gouraud triangle (2face)
GsTMDdivG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsTMDdivG4LB	TMD_P_G4	Gouraud quadrilateral (light source calculation + 2face)
GsTMDdivG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsTMDdivG4LFGB	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDdivG4NL	TMD_P_G4	Gouraud quadrilateral
GsTMDdivG4NLB	TMD_P_G4	Gouraud quadrilateral (2face)
GsTMDdivTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsTMDdivTF3LB	TMD_P_TF3	Textured flat triangle (light source calculation + 2face)
GsTMDdivTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsTMDdivTF3LFGB	TMD_P_TF3	Textured flat triangle (light source calculation +FOG + 2face)
GsTMDdivTF3NL	TMD_P_TF3	Textured flat triangle
GsTMDdivTF3NLB	TMD_P_TF3	Textured flat triangle (2face)
GsTMDdivTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsTMDdivTF4LB	TMD_P_TF4	Textured flat quadrilateral (light source calculation + 2face)
GsTMDdivTF4LM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +mip-map)
GsTMDdivTF4LFG	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG)
GsTMDdivTF4LFGB	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG + 2face)
GsTMDdivTF4LFGM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG+mip-map)

Low-level function name	First arg (op) type	Description
GsTMDdivTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsTMDdivTF4NLB	TMD_P_TF4	Textured flat quadrilateral (2face)
GsTMDdivTF4NLM	TMD_P_TF4	Textured flat quadrilateral (mip-map)
GsTMDdivTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsTMDdivTG3LB	TMD_P_TG3	Textured Gouraud triangle (light source calculation + 2face)
GsTMDdivTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsTMDdivTG3LFGB	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG + 2face)
GsTMDdivTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsTMDdivTG3NLB	TMD_P_TG3	Textured Gouraud triangle (2face)
GsTMDdivTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsTMDdivTG4LB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + 2face)
GsTMDdivTG4LM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +mip-map)
GsTMDdivTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsTMDdivTG4LFGB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDdivTG4LFGM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG + mip-map)
GsTMDdivTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsTMDdivTG4NLB	TMD_P_TG4	Textured Gouraud quadrilateral (2face)
GsTMDdivTG4NLM	TMD_P_TG4	Textured Gouraud quadrilateral (mip-map)

Table 9-10: GsTMDdivN...() [no normals]

Low-level function name	First arg (op) type	Description
GsTMDdivNF3	TMD_P_NF3	Flat triangle
GsTMDdivNF3B	TMD_P_NF3	Flat triangle (2face)
GsTMDdivNF4	TMD_P_NF4	Flat quadrilateral
GsTMDdivNF4B	TMD_P_NF4	Flat quadrilateral (2face)
GsTMDdivNG3	TMD_P_NG3	Gouraud triangle
GsTMDdivNG3B	TMD_P_NG3	Gouraud triangle (2face)
GsTMDdivNG4	TMD_P_NG4	Gouraud quadrilateral
GsTMDdivNG4B	TMD_P_NG4	Gouraud quadrilateral (2face)
GsTMDdivTNF3	TMD_P_TNF3	Textured flat triangle
GsTMDdivTNF3B	TMD_P_TNF3	Textured flat triangle (2face)
GsTMDdivTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsTMDdivTNF4B	TMD_P_TNF4	Textured flat quadrilateral (2face)
GsTMDdivTNF4M	TMD_P_TNF4	Textured flat quadrilateral (mip-map)
GsTMDdivTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsTMDdivTNG3B	TMD_P_TNG3	Textured Gouraud triangle (2face)
GsTMDdivTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral
GsTMDdivTNG4B	TMD_P_TNG4	Textured Gouraud quadrilateral (2face)
GsTMDdivTNG4M	TMD_P_TNG4	Textured Gouraud quadrilateral (mip-map)

Number of divisions is specified in the GsDOBJ2 attribute.

Return value

Starting address of unused packet area.

See also

[GsSortObject4J\(\)](#)

GsTMDfast..., GsTMDfastN...

Low-level functions for GsSortObject4J().

Library	Header File	Introduced	Documentation Date
<i>libgte.lib</i>	<i>libgte.h</i>	4.1	12/14/98

Syntax

PACKET *GsTMDfast...(
TMD_P_... *op, Pointer to starting address of TMD data primitives
VERT *vp, Pointer to starting address of TMD data vertices TMD
VERT *np, Pointer to starting address of TMD data normals
PACKET *pk, Pointer to top address of GPU packet buffer
int n, Number of primitives
int shift, OT shift bit
GsOT *ot, Pointer to GsOT
u_long *scratch) Pointer to starting address of unused scratch pad

PACKET *GsTMDfastN...(
TMD_P_... *op, Pointer to starting address of TMD data primitives
VERT *vp, Pointer to starting address of TMD data vertices TMD
PACKET *pk, Pointer to top address of GPU packet buffer
int n, Number of primitives
int shift, OT shift bit
GsOT *ot, Pointer to GsOT
u_long *scratch) Pointer to starting address of unused scratch pad

Explanation

These are low-level functions for GsSortObject4J(). They are tuned assembly-level functions for high speed operation in libgte.

To be used, they must be registered in GsFCALL4 as low-level functions.

These functions perform coordinate and perspective transformation, backface clipping, and light source calculation for *n* primitives, complete the GPU packet in the buffer, and link it into the OT.

For low-level functions that provide material attenuation, the lighting mode must be "material ON." This attenuates the brightness during light source calculation based on the parameter which is provided in the attribute of GsDOBJ2.

For low-level functions that FLIP, the direction of the normal clip is reversed.

For function types which do not operate on normals within the data (e.g. GsTMDfastN...), light source calculations are not performed so fewer parameters are passed compared to those function types which operate on normals (e.g. GsTMDfast...),

Low-level functions supported in libgs are shown below.

Table 9-11: GsTMDfast...() [have normals]

Low-level function name	First arg (op) type	Description
GsTMDfastF3L	TMD_P_F3	Flat triangle (light source calculation)
GsTMDfastF3LB	TMD_P_F3	Flat triangle (light source calculation + 2face)
GsTMDfastF3LFG	TMD_P_F3	Flat triangle (light source calculation +FOG)
GsTMDfastF3LFGB	TMD_P_F3	Flat triangle (light source calculation +FOG + 2face)
GsTMDfastF3NL	TMD_P_F3	Flat triangle
GsTMDfastF3NLB	TMD_P_F3	Flat triangle (2face)
GsTMDfastF4L	TMD_P_F4	Flat quadrilateral (light source calculation)
GsTMDfastF4LB	TMD_P_F4	Flat quadrilateral (light source calculation + 2face)
GsTMDfastF4LFG	TMD_P_F4	Flat quadrilateral (light source calculation +FOG)
GsTMDfastF4LFGB	TMD_P_F4	Flat quadrilateral (light source calculation +FOG + 2face)
GsTMDfastF4NL	TMD_P_F4	Flag quadrilateral
GsTMDfastF4NLB	TMD_P_F4	Flag quadrilateral (2face)
GsTMDfastG3L	TMD_P_G3	Gouraud triangle (light source calculation)
GsTMDfastG3LB	TMD_P_G3	Gouraud triangle (light source calculation + 2face)
GsTMDfastG3LFG	TMD_P_G3	Gouraud triangle (light source calculation +FOG)
GsTMDfastG3LFGB	TMD_P_G3	Gouraud triangle (light source calculation +FOG + 2face)
GsTMDfastG3NL	TMD_P_G3	Gouraud triangle
GsTMDfastG3NLB	TMD_P_G3	Gouraud triangle (2face)
GsTMDfastG4L	TMD_P_G4	Gouraud quadrilateral (light source calculation)
GsTMDfastG4LB	TMD_P_G4	Gouraud quadrilateral (light source calculation + 2face)
GsTMDfastG4LFG	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG)
GsTMDfastG4LFGB	TMD_P_G4	Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDfastG4NL	TMD_P_G4	Gouraud quadrilateral
GsTMDfastG4NLB	TMD_P_G4	Gouraud quadrilateral (2face)
GsTMDfastTF3L	TMD_P_TF3	Textured flat triangle (light source calculation)
GsTMDfastTF3LB	TMD_P_TF3	Textured flat triangle (light source calculation + 2face)
GsTMDfastTF3LFG	TMD_P_TF3	Textured flat triangle (light source calculation +FOG)
GsTMDfastTF3LFGB	TMD_P_TF3	Textured flat triangle (light source calculation +FOG + 2face)
GsTMDfastTF3NL	TMD_P_TF3	Textured flat triangle
GsTMDfastTF3NLB	TMD_P_TF3	Textured flat triangle (2face)
GsTMDfastTF4L	TMD_P_TF4	Textured flat quadrilateral (light source calculation)
GsTMDfastTF4LB	TMD_P_TF4	Textured flat quadrilateral (light source calculation + 2face)
GsTMDfastTF4LM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +mip-map)

Low-level function name	First arg (op) type	Description
GsTMDfastTF4LFG	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG)
GsTMDfastTF4LFGB	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG + 2face)
GsTMDfastTF4LFGM	TMD_P_TF4	Textured flat quadrilateral (light source calculation +FOG+mip-map)
GsTMDfastTF4NL	TMD_P_TF4	Textured flat quadrilateral
GsTMDfastTF4NLB	TMD_P_TF4	Textured flat quadrilateral (2face)
GsTMDfastTF4NLM	TMD_P_TF4	Textured flat quadrilateral (mip-map)
GsTMDfastTG3L	TMD_P_TG3	Textured Gouraud triangle (light source calculation)
GsTMDfastTG3L_FLIP	TMD_P_TG3	Textured Gouraud triangle (light source calculation + FLIP)
GsTMDfastTG3LB	TMD_P_TG3	Textured Gouraud triangle (light source calculation + 2face)
GsTMDfastTG3LFG	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG)
GsTMDfastTG3LFG_FLIP	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG + FLIP)
GsTMDfastTG3LFGB	TMD_P_TG3	Textured Gouraud triangle (light source calculation +FOG + 2face)
GsTMDfastTG3NL	TMD_P_TG3	Textured Gouraud triangle
GsTMDfastTG3NL_FLIP	TMD_P_TG3	Textured Gouraud triangle (FLIP)
GsTMDfastTG3NLB	TMD_P_TG3	Textured Gouraud triangle (2face)
GsTMDfastTG4L	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation)
GsTMDfastTG4LB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation + 2face)
GsTMDfastTG4LM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +mip-map)
GsTMDfastTG4LFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG)
GsTMDfastTG4LFGB	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG + 2face)
GsTMDfastTG4LFGM	TMD_P_TG4	Textured Gouraud quadrilateral (light source calculation +FOG + mip-map)
GsTMDfastTG4NL	TMD_P_TG4	Textured Gouraud quadrilateral
GsTMDfastTG4NLB	TMD_P_TG4	Textured Gouraud quadrilateral (2face)
GsTMDfastTG4NLM	TMD_P_TG4	Textured Gouraud quadrilateral (mip-map)
GsTMDfastF3GL	TMD_P_F3G	Flat gradation triangle (light source calculation)
GsTMDfastF3GLFG	TMD_P_F3G	Flat gradation triangle (light source calculation + FOG)
GsTMDfastF3GNL	TMD_P_F3G	Flat gradation triangle
GsTMDfastG3GL	TMD_P_G3G	Gouraud gradation triangle (light source calculation)
GsTMDfastG3GLFG	TMD_P_G3G	Gouraud gradation triangle (light source calculation + FOG)
GsTMDfastG3GNL	TMD_P_G3G	Gouraud gradation triangle
GsTMDfastF4GL	TMD_P_F4G	Flat gradation quadrilateral (light source calculation)
GsTMDfastF4GLFG	TMD_P_F4G	Flat gradation quadrilateral (light source calculation + FOG)
GsTMDfastF4GNL	TMD_P_F4G	Flat gradation quadrilateral
GsTMDfastG4GL	TMD_P_G4G	Gouraud gradation quadrilateral (light source calculation)
GsTMDfastG4GLFG	TMD_P_G4G	Gouraud gradation quadrilateral (light source calculation + FOG)

Low-level function name	First arg (op) type	Description
GsTMDfastG4GNL	TMD_P_G4G	Gouraud gradation quadrilateral
GsTMDfastF3M	TMD_P_F3	Flat triangle (light source + material attenuation)
GsTMDfastF3MFG	TMD_P_F3	Flat triangle (light source + FOG + material attenuation)
GsTMDfastF4M	TMD_P_F4	Flat quadrilateral (light source + material attenuation)
GsTMDfastF4MFG	TMD_P_F4	Flat quadrilateral (light source + FOG + material attenuation)
GsTMDfastG3M	TMD_P_G3	Gouraud triangle (light source + material attenuation)
GsTMDfastG3MFG	TMD_P_G3	Gouraud triangle (light source + FOG + material attenuation)
GsTMDfastG4M	TMD_P_G4	Gouraud quadrilateral (light source + material attenuation)
GsTMDfastG4MFG	TMD_P_G4	Gouraud quadrilateral (light source + FOG + material attenuation)
GsTMDfastTF3M	TMD_P_TF3	Textured flat triangle (light source + material attenuation)
GsTMDfastTF3MFG	TMD_P_TF3	Textured flat triangle (light source + FOG + material attenuation)
GsTMDfastTF4M	TMD_P_TF4	Textured flat quadrilateral (light source + material attenuation)
GsTMDfastTF4MFG	TMD_P_TF4	Textured flat quadrilateral (light source + FOG + material attenuation)
GsTMDfastTG3M	TMD_P_TG3	Textured Gouraud triangle (light source + material attenuation)
GsTMDfastTG3MFG	TMD_P_TG3	Textured Gouraud triangle (light source + FOG + material attenuation)
GsTMDfastTG4M	TMD_P_TG4	Textured Gouraud quadrilateral (light source + material attenuation)
GsTMDfastTG4MFG	TMD_P_TG4	Textured Gouraud quadrilateral (light source + FOG + material attenuation)

Table 9-12: GsTMDfastN...() [no normals]

Low-level function name	First arg (op) type	Description
GsTMDfastNF3	TMD_P_NF3	Flat triangle
GsTMDfastNF3B	TMD_P_NF3	Flat triangle (2face)
GsTMDfastNF4	TMD_P_NF4	Flat quadrilateral
GsTMDfastNF4B	TMD_P_NF4	Flat quadrilateral (2face)
GsTMDfastNG3	TMD_P_NG3	Gouraud triangle
GsTMDfastNG3B	TMD_P_NG3	Gouraud triangle (2face)
GsTMDfastNG4	TMD_P_NG4	Gouraud quadrilateral
GsTMDfastNG4B	TMD_P_NG4	Gouraud quadrilateral (2face)
GsTMDfastTNF3	TMD_P_TNF3	Textured flat triangle
GsTMDfastTNF3B	TMD_P_TNF3	Textured flat triangle (2face)
GsTMDfastTNF4	TMD_P_TNF4	Textured flat quadrilateral
GsTMDfastTNF4B	TMD_P_TNF4	Textured flat quadrilateral (2face)
GsTMDfastTNF4M	TMD_P_TNF4	Textured flat quadrilateral (mip-map)
GsTMDfastTNG3	TMD_P_TNG3	Textured Gouraud triangle
GsTMDfastTNG3_FLIP	TMD_P_TNG3	Textured Gouraud triangle (FLIP)
GsTMDfastTNG3B	TMD_P_TNG3	Textured Gouraud triangle (2face)
GsTMDfastTNG4	TMD_P_TNG4	Textured Gouraud quadrilateral
GsTMDfastTNG4B	TMD_P_TNG4	Textured Gouraud quadrilateral (2face)
GsTMDfastTNG4M	TMD_P_TNG4	Textured Gouraud quadrilateral (mip-map)

With gradation, each vertex of the TMD polygon has a different RGB value.

Return value

Starting address of unused packet area.

See also

[GsSortObject4J\(\)](#)

Macros

GsClearDispArea

Clears screen.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.0	12/14/98

Syntax

GsClearDispArea(
u_char *r*, **u_char** *g*, **u_char** *b*) Background color RGB values

Explanation

The display area is cleared using IO.

Unlike GsSortClear(), a clear command is issued when GsClearDispArea() is called.

See also

[GsSortClear\(\)](#)

GsIncFrame

Updates the frame ID.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.2	12/14/98

Syntax

GsIncFrame()

Explanation

GsIncFrame() is a macro called from within GsSwapDispBuff(). It increments the global variable PSDCNT by 1. PSDCNT is 32 bits in length, and restarts at 1 rather than 0 when it overflows.

PSDCNT is used by GsGetLw(),GsGetLs(),GsGetLws() when determining the validity of the matrix cache.

If you are not using GsSwapDispBuff() to swap double buf you must call GsIncFrame to swap the buffers when you use GsGetLw(), GsGetLs(), and GsGetLws().

Use GsDefDispBuff() to establish settings the first time.

See also

[GsGetLw\(\)](#), [GsGetLs\(\)](#), [GsGetLws\(\)](#), [GsSwapDispBuff\(\)](#)

GsSetAzwh

Sets conditions for active subdivision.

Library	Header File	Introduced	Documentation Date
<i>libgs.lib</i>	<i>libgs.h</i>	3.3	12/14/98

Syntax

void GsSetAzwh(

int z,

short w, short h)

Critical near z value for activate subdivision

Size of polygon within subdivision routine at which no more subdivision will be done

Explanation

Sets the conditions for active subdivision.

z is the near z value for the start of the subdivision fragmentation and w, h is the polygon size for halting the subdivision.

See also

[GsA4Div...\(\)](#)

External Variables

Table 9-13: List of External Variables

Name of external variable	Type	Description
CLIP2	RECT	Two-dimensional clipping area. Can be set with GsSetClip2D()
PSDBASEX[2]	short	Double buffer origin (X coordinate) Set with GsDefDispbuff()
PSDBASEY[2]	short	Double buffer origin (Y coordinate) Set with GsDefDispbuff()
PSDIDX	short	Double buffer index
PSDCNT	u_long	Incremented with each frame switch
POSITION	_GsPOSITION	Two-dimensional offset
GsDRAWENV	DRAWENV	Gs draw environment
GsDISPENV	DISPENV	Gs display environment
GsWSMATRIX	MATRIX	Gs world-to-screen matrix Set with GsSetRefView2(), etc.
GsLIGHT_MODE	int	Default light mode
HWD0	long	Horizontal resolution
VWD0	long	Vertical resolution
GsLIGHTWSMATRIX	MATRIX	Gs light matrix. Set with GsSetFlatLight()
GsIDMATRIX	MATRIX	Unit matrix
GsIDMATRIX2	MATRIX	Unit matrix (includes aspect conversion)
GsLIGHT_FUNC	Function pointer	Pointer to default light-source calculation routines used by GsDOBJ1, GsDOBJ2
GsOUT_PACKET_P	u_long	Pointer for saving start of packet area. Set with GsSetWorkBase()
GsMATE_C	u_long	Result of decoding attribute (attenuation coefficient)
GsLMODE	u_long	Result of decoding attribute (Light mode)
GsLIGNR	u_long	Result of decoding attribute (Ignore light)
GsLIOFF	u_long	Result of decoding attribute (No shading)
GsZOVER	u_long	Result of decoding attribute (nearZ)
GsBACKC	u_long	Result of decoding attribute (Two-sided polygons)
GsNDIV	u_long	Result of decoding attribute (Number of divisions)
GsTRATE	u_long	Result of decoding attribute (Semi-transparency rate)
GsTON	u_long	Result of decoding attribute (Semi-transparency)
GsDISPON	u_long	Result of decoding attribute (Display ON/OFF)
GsADIVH	short	Condition for active automatic divisions: Vertical size Set with GsSetAzwh(z,w,h)
GsADIVW	short	Condition for active automatic divisions: Horizontal size Set with GsSetAzwh(z,w,h)

Name of external variable	Type	Description
GsADIVZ	long	Condition for active automatic divisions: Z value Set with GsSetAzwh(z,w,h)
GsFCALL5	Structure	Function table for GsSortObject5J()
GsFCALL4	Structure	Function table for GsSortObject4J()

Chapter 10: CD/Streaming Library

Table of Contents

Structures

CdIATV	10-3
CdIFILE	10-4
CdIFILTER	10-5
CdILOC	10-6
StHEADER	10-7

Functions

CdComstr	10-8
CdControl	10-9
CdControlB	10-11
CdControlF	10-12
CdDataCallback	10-13
CdDataSync	10-14
CdDiskReady	10-15
CdFlush	10-16
CdGetDiskType	10-17
CdGetSector	10-18
CdGetSector2	10-19
CdGetToc	10-20
CdInit	10-21
CdIntstr	10-22
CdIntToPos	10-23
CdLastCom	10-24
CdLastPos	10-25
CdMix	10-26
CdMode	10-27
CdPlay	10-28
CdPosToInt	10-29
CdRead	10-30
CdRead2	10-31
CdReadBreak	10-32
CdReadCallback	10-33
CdReadExec	10-34
CdReadFile	10-35
CdReadSync	10-36
CdReady	10-37
CdReadyCallback	10-38
CdReset	10-39
CdSearchFile	10-40
CdSetDebug	10-41
CdStatus	10-42
CdSync	10-43
CdSyncCallback	10-44
StCdInterrupt	10-45
StClearRing	10-46
StFreeRing	10-47
StGetBackloc	10-48
StGetNext	10-49
StGetNextS	10-50
StNextStatus	10-51
StRingStatus	10-52
StSetChannel	10-53
StSetEmulate	10-54

StSetMask	10-55
StSetRing	10-56
StSetStream	10-57
StUnSetRing	10-58

CdIATV

Audio attenuation structure.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	8/9/99

Structure

```
typedef struct {
    u_char val0;      CD (L) --> SPU (L)
    u_char val1;      CD (L) --> SPU (R) (CD left sound transferred to right)
    u_char val2;      CD (R) --> SPU (R)
    u_char val3;      CD (R) --> SPU (L) (CD right sound transferred to left)
} CdIATV;
```

Explanation

Sets CD audio volume (consisting of CD-DA audio and CD-XA audio).

val0 - *val3* can range from 0 (minimum volume) to 128 (maximum volume).

For adjusting normal stereo volume, set the L channel volume in *val0* and the R channel volume in *val2*. *val1* and *val3* should be set to 0.

See also

[CdMix\(\)](#)

CdIFILE

ISO-9660 file system file descriptor.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Structure

```
typedef struct {  
    CdILOffset pos;           File position  
    u_long size;             File size  
    char name[16];          File name  
} CdIFILE;
```

Explanation

Position and size of ISO-9660 CD-ROM file.

See also

[CdSearchFile\(\)](#)

CdIFILTER

ADPCM channel.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    u_char file;      File ID
    u_char chan;      Channel ID
    u_short pad;       System reserved
} CdIFILTER;
```

Explanation

Defines a multi-channel ADPCM play channel. Use CdISetfilter command of CdControl() to set.

See also

[CdControl\(\)](#)

CdILOC

Time-code based CD-ROM disc position.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    u_char minute;      Minute
    u_char second;      Second
    u_char sector;      Sector
    u_char track;       Track number
} CdILOC;
```

Explanation

Defines a time-code position on a CD-ROM, based on the time needed to reach that position when playing the disc from the beginning at normal speed.

The *track* member is not used at present.

Use CdISetloc command of CdControl() to set.

See also

[CdControl\(\)](#)

StHEADER

Sector header.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    u_short id;           Reserved by system
    u_short type;         Data type (always 0x0160)
    u_short secCount;     Sector offset within 1 frame
    u_short nSectors;     Number of sectors comprising one frame
    u_long frameCount;    Movie absolute frame number
    u_long frameSize;     Movie data size (in long words)
    u_short width;        Movie horizontal size
    u_short height;       Movie vertical size
    u_long dummy1;        Reserved by system
    u_long dummy2;        Reserved by system
    CdILOC loc;           File location
} StHEADER;
```

Explanation

Movie sector header. If a header obtained with `StGetNext()` is written to this structure, the data can be accessed through the structure members.

For details of the structure, refer to “STR: Streaming (Movie) Data” in the File Formats document.

CdComstr

Get character string corresponding to command code (for debugging).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
char *CdComstr(  
u_char com)           Command completion code
```

Explanation

For debugging. Get corresponding character string from processing status code. Example: CdlNop returns “CdlNop”, CdlSetloc returns “CdlSetloc”, and so on.

Return value

Pointer to start of character string.

See also

[CdlIntstr\(\)](#), [CdSetDebug\(\)](#)

CdControl

Issue primitive command to CD-ROM system.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdControl(
    u_char com,           Command code
    u_char *param,        Pointer to command arguments
    u_char *result)       Pointer to return value storage buffer (requires 8 bytes)
```

Explanation

Issues the primitive command *com* to the CD-ROM system. *param* points to the arguments of the command, if any; set *param* to 0 for commands that do not require arguments. *result* points to a buffer used to hold the return value; if *result* is 0, the return value is not stored.

The values of command (*com*), arguments (*param*), and return value (*result*) are listed below. For the functions that are non-blocking, the actual transmission completion must be detected by CdSync()

Table 10–1: Primitive commands

Symbol	Code	Type of Operation	Contents
CdINop	0x01	Blocking	NOP (No Operation)
CdISetloc	0x02	Blocking	Set the seek target position
CdIPlay	0x03	Blocking	Commence CD-DA play
CdIForward	0x04	Blocking	Forward
CdIBackward	0x05	Blocking	Rewind
CdIReadN	0x06	Blocking	Start data read (with retry)
CdIStandby	0x07	Nonblocking	Stand by with disk rotating
CdIStop	0x08	Nonblocking	Disk stopped
CdIPause	0x09	Nonblocking	Pause at current position
CdIMute	0x0b	Blocking	CD-DA mute
CdIDemute	0x0c	Blocking	Cancel mute
CdISetfilter	0x0d	Blocking	Choose ADPCM play sector
CdISetmode	0x0e	Blocking	Set basic mode (see Table 10-4)
CdIGetparam	0x0f	Blocking	Gets current CD subsystem mode
CdIGetlocL	0x10	Blocking	Get logical location (data sector)
CdIGetlocP	0x11	Blocking	Get physical location (audio sector)
CdISseekL	0x15	Nonblocking	Logical seek (data sector seek)
CdISseekP	0x16	Nonblocking	Physical seek (audio sector seek)
CdIReadS	0x1b	Blocking	Commence data read (no retry)

Table 10–2: Primitive commands that take arguments and their arguments

Symbol	Argument Type	Contents
CdlSetloc	CdlLOC	Start sector location
CdlReadN	CdlLOC	Start sector location
CdlReadS	CdlLOC	Start sector location
CdlPlay	CdlLOC	Start sector location
CdlSetfilter	CdlFILTER	Set ADPCM sector play
CdlSetmode	u_char	Set basic mode

Table 10–3: Return values of primitive commands

Return Value and Stored Byte Position								
Symbol	0	1	2	3	4	5	6	7
CdlGetparam	Status	Mode						
CdlGetlocL	Min	Sec	Sector	Mode	File	Chan		
CdlGetlocP	Track	Index	Min	Sec	Frame	Amin	Asec	Aframe
CdlSeekL	Status	Btrack	Etrack					
CdlSeekP	Status	Min	Sec					

Note: all other commands return Status in the first byte.

Return value

1 if the command is issued successfully; 0 if failed.

See also

[CdControlB\(\)](#), [CdControlF\(\)](#)

CdControlB

Issue primitive command to CD-ROM system (Blocking-type function).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdControlB(
  u_char com,           Command code
  u_char *param,        Pointer to command arguments
  u_char *result)       Pointer to return value storage buffer (requires 8 bytes)
```

Explanation

Issues the primitive command *com* to the CD-ROM system. *param* points to the arguments of the command, if any; set *param* to 0 for commands that do not require arguments. *result* points to a buffer used to hold the return value; if *result* is 0, the return value is not stored.

CdControlB() is identical to CdControl() except that it is blocking; that is, it waits for all commands to terminate before returning. For details, see CdControl().

Return value

1 if issued successfully; 0 if failed.

See also

[CdControl\(\)](#), [CdControlF\(\)](#)

CdControlF

Issue primitive command to CD-ROM system (high speed).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdControlF(
  u_char com,           Command code (see separate item)
  u_char *param)        Pointer to an argument for command
```

Explanation

Issues the primitive command *com* to the CD-ROM system. *param* points to the arguments of the command, if any; set *param* to 0 for commands that do not require arguments. *result* points to a buffer used to hold the return value; if *result* is 0, the return value is not stored.

CdControlF() is fast because it does no handshaking with the subsystem (it does not even wait for command acknowledgement (ACK)). For details, see the commands and arguments of CdControl(), and the Run-time Library Overview.

Return value

Always returns 1.

See also

[CdControl\(\)](#), [CdControlB\(\)](#)

CdDataCallback

Define a routine to be executed when data transfer is completed.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.0	12/14/98

Syntax

```
void *CdDataCallback (
void (*func) ())      Pointer to address of callback function
```

Explanation

Defines a routine *func* to be executed when the data transfer initiated by CdGetSector() or CdGetSector2() has been completed. If *func* is 0, then any previous callback routine is disabled.

While *func* is executing, subsequent data transfer complete interrupts are masked. Therefore, *func* should return as soon as the necessary processing is completed.

Return Value

Address of previously set callback

See also

[CdGetSector\(\)](#), [CdGetSector2\(\)](#), [CdDataSync\(\)](#)

CdDataSync

Wait for or check a data transfer initiated by CdGetSector2().

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.2	12/14/98

Syntax

```
int CdDataSync(  
int mode)           Polling mode
```

Explanation

If *mode* is 0, the function waits for a data transfer initiated by CdGetSector2() to be completed. If *mode* is 1, the function polls the current status and returns.

Return Value

0 if transfer is completed; 1 if transfer is still being performed; -1 if an error occurred.

See also

[CdGetSector2\(\)](#), [CdDataCallback\(\)](#)

CdDiskReady

Determine CD-ROM status after disc change.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

```
int CdDiskReady(
int mode)           0: Blocking type, 1: Non-blocking type
```

Explanation

Checks the CD-ROM status after a disc change to determine whether a command can be issued safely. (Immediately after a disc is changed, there is a delay of a few seconds during which commands may not be issued.)

When *mode* is 0, this function waits until the CD-ROM status has stabilized and commands may be issued before returning. When the *mode* parameter is 1, this function simply returns the current status.

It is recommended that your program use this function immediately after initiating a disc change to wait for the disc cover to be closed and the CD-ROM status to stabilize. After this is done, check the disc format using the `CDGetDiskType()` and proceed accordingly.

The maximum wait time required for returning from a blocking type function call is:

DebuggingStation:

CD-R	Maximum of 12 seconds
CD-DA	Maximum of 12 seconds
No disc	Approximately 5 seconds

PlayStation®:

Black CD	Maximum of 3 seconds
CD-DA	Maximum of 5 seconds
No disc	Approximately 5 seconds

Although non-blocking type function returns immediately after checking the disc status, it cannot differentiate two error cases, the non-stable status and no disc case. Thus it is recommended to manage the time out according to the wait time shown above.

Note: This function does not operate correctly on the DTL-H2000 development system.

Return value

<code>CdlComplete</code>	The state where a command can be issued
<code>CdlDiskError</code>	Blocking type: No discs or defected disc Non-blocking type: Not stable, no discs, or defected disc
<code>CdlStatShellOpen</code>	Disc cover is open

CdFlush

CD-ROM command flush.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.2	12/14/98

Syntax

void CdFlush(void)

Explanation

Cancels the CD-ROM subsystem command being issued. The command being issued is invalidated and the subsystem can immediately accept commands.

Since the data receipt mode is reset to CdlDataReady, be careful not to use this function when reading from the CD-ROM.

CdGetDiskType

Get disc format.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	2/24/99

Syntax

```
int CdGetDiskType(void)
```

Explanation

Obtains the disc format. Currently only CD-ROM format can be recognized.

On DebuggingStation, although PlayStation disc (black disc), CD-R, and other CD-ROM (ISO9600 format) can be recognized as a CD-ROM, on PlayStation (consumer model), only the PlayStation disc can be recognized as CD-ROM. CD-DA is always recognized as "Other Format".

Note: Immediately after changing discs, it is recommended that your program call CdDiskReady(), followed by CdGetDiskType().

Since this function internally issues the CdISetmode command and sets the mode to CdIModeSpeed, the mode must be reset as necessary after calling the function.

For details, refer to the CdGetDiskType() function in sample\cd\tuto\tuto11.c.

Note: This function does not operate correctly on the DTL-H2000 development system.

Return value

CdICdromFormat	CD-ROM format
CdIOtherFormat	Other format
CdIStatShellOpen	Disc cover is open
CdIStatNoDisk	No discs

CdGetSector

Transfer sector buffer data to main memory.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

int CdGetSector(

void *madr,

int size)

Main memory address to which the data is transferred

Size of data to be transferred (long words)

Explanation

Transfers data in the sector buffer to main memory. Transferring is processed in background.

Sector size differs from mode to mode. See Table 10-4 for a list of modes.

Although the data transfer can be divided several times to different memory addresses, the sector data must be read after the buffer becomes ready and before the buffer is overwritten by the next sector data. This function blocks until data transfer is complete.

If less than a full sector is transferred during the callback, it is no longer necessary to issue dummy reads to update the pointer to the next sector. (In previous versions of CdGetSector() it was necessary to read excess data into a dummy area of RAM.)

Return Value

Always returns 1.

See also

[CdDataSync\(\)](#), [CdDataCallback\(\)](#), [CdGetSector2\(\)](#)

CdGetSector2

Transfer sector buffer data to main memory (Non-blocking type).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	4.0	12/14/98

Syntax

```
int CdGetSector2(
void *madr,           Main memory address to which data is transferred
int size)             Size of data to be transferred (long words)
```

Explanation

Transfers sector buffer data to main memory. Since the transfer is carried out in cycle steal mode (a DMA mode that allows more sharing of the bus with other devices), an interrupt can be inserted within the transfer. Because this function is non-blocking, transfer completion must be monitored by CdDataSync() or CdDataCallback().

Furthermore, although an interrupt can be inserted during transfer in cycle steal mode, since a normal program cannot use a data bus, if there is a command to access the memory through a bus, processing is blocked at that point.

Data transfer in cycle steal mode takes longer than in block mode (the mode used by CdGetSector()).

If less than a full sector is transferred during the callback, it is no longer necessary to issue dummy reads to update the pointer to the next sector. (In previous versions of CdGetSector2() it was necessary to read excess data into a dummy area of RAM.)

Return value

Always returns 1.

See also

[CdDataCallback\(\)](#), [CdDataSync\(\)](#), [CdGetSector\(\)](#).

CdGetToc

Read CD-ROM table of contents information.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdGetToc(  
CdLOC *loc)           Pointer to location table
```

Explanation

Get starting position of each track on the CD-ROM disc.

The maximum number of tracks is 100.

Return value

Positive integer is a track number; anything else is an error.

See also

[CdSearchFile\(\)](#)

CdInit

Initialize CD-ROM system.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
void CdInit(  
CdInit mode)           Reset mode (use 0)
```

Explanation

Resets the CD-ROM subsystem. The *mode* parameter is not used by current versions of the library and should be set to 0.

When initialization fails, up to four retries are performed. Since CdInit() and CdReset() reset the SPU sound volume and CD input volume to the SPU, etc., they must be called before libspu/libsnd initialization and setting functions.

Return value

1 if initialization is successful; 0 on failure.

See also

[CdReset\(\)](#)

CdIntstr

Get character string corresponding to command status code (for debugging).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
char *CdIntstr(  
u_char intr)           Processing status code
```

Explanation

For debugging. Get character string corresponding to processing status code *intr*. For debugging.

For example, CdINolntr returns "CdINolntr", and so on.

Return value

Pointer to start of character string

See also

[CdComstr\(\)](#), [CdSetDebug\(\)](#)

CdIntToPos

Translate an absolute sector number to a minute/seconds/sector time code.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

[CdILOC](#) *CdIntToPos(

[int](#) *i*, Absolute sector number

[CdILOC](#) **p*) Pointer to a CdILOC structure that will be set to the position time code

Explanation

Calculate value for minute/second/sector from absolute sector number.

Return value

p

See also

[CdPosToInt\(\)](#)

CdLastCom

Get last command issued by CDControl()/CDControlB()

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.2	12/14/98

Syntax

int CdLastCom(*void*)

Explanation

Returns the last command issued by CDControl()/CDControlB().

See Table 10–1 under CDControl() for a list of the commands.

Return Value

Last command.

See also

[CdControl\(\)](#), [CdControlB\(\)](#)

CdLastPos

Get CD-ROM location most recently specified.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

CdILOC *CdLastPos(void)

Explanation

Returns the latest location specified by one of the sub commands CdISetloc/CdIPlay/CdISeekL/CdISeekP/CdIRead/CdIReadS.

Return value

Pointer to the CdILOC structure containing the CD-ROM location.

CdMix

Set attenuation for CD audio.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdMix(  
CdIAudio *vol)           Pointer to attenuator volume
```

Explanation

Set audio volume value for CD audio (CD-DA, ADPCM).

Return value

1.

CdMode

Get latest CD-ROM mode.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

int CdMode(void)

Explanation

Returns the latest CD-ROM mode set.

High speed since this function only refers to the status in the main memory. The mode buffer is updated when a CdISetmode command is issued and also when the mode is specified in the arguments as in CdRead(), etc.

Table 10-4: CD Mode Settings

Symbol	Code	Details		
CdIModeStream	0x100	Normal streaming		
CdIModeStream2	0x120	SUB HEADER information includes		
CdIModeSpeed	0x80	Transfer speed	0: Normal speed	1: Double speed
CdIModeRT	0x40	ADPCM play	0: ADPCM OFF	1: ADPCM ON
CdIModeSize1	0x20	Sector size	0: 2048 byte	1: 2340byte
CdIModeSize0	0x10	Sector size	0: —	1: 2328byte
CdIModeSF	0x08	Subheader filter	0: Off	1: On
CdIModeRept	0x04	Report mode	0: Off	1: On
CdIModeAP	0x02	Autopause	0: Off	1: On
CdIModeDA	0x01	CD-DA play	0: CD-DA off	1: CD-DA on

Return value

CD-ROM mode.

See also

[CdRead\(\)](#)

CdPlay

Play CD-DA tracks.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

int CdPlay(

int mode,

0: Stops playing

1: Plays tracks in *tracks* array in the specified order. Stop at end.

2: Plays tracks in *tracks* array in the specified order. Repeat at end.

3: Returns an index of the array corresponding to the track currently being played.

int *tracks,

Pointer to array specifying the tracks to be played. Must end with 0.

int offset)

Index of the tracks to be played.

Explanation

Plays multiple tracks specified by the array *tracks* in order. After playing the last track of the array, it repeats or stops playing according to the mode specified.

All playing is done in units of tracks. Playing or stopping in the middle of a track is not allowed.

Return value

Index of the track currently being played (not the track number). -1 when it has already stopped playing.

CdPosToInt

Translate time code to an absolute sector number.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

int CdPosToInt(
[CdILOC *p](#)) Pointer to a CdILOC structure containing the position time code

Explanation

Translates a minutes/seconds/sectors time code contained in a cdILOC structure pointed to by the *p* parameter into an absolute sector value.

Return value

Absolute sector number

See also

[CdIntToPos\(\)](#)

CdRead

Read multiple sectors from the CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdRead(
    int sectors,           Read sector count
    u_long *buf,          Pointer to read buffer
    int mode)             CD-ROM subsystem mode, as defined for CdISetmode command of
                          CdControl()
```

Explanation

Reads one or more sectors of data from the CD-ROM to the specified buffer in memory. The starting position for the read is the position last specified for CdISetloc. Each CdRead() requires a separate CdISetloc command.

CdRead() is non-blocking. Check for completion using CdReadSync() or CdReadCallback(). CdRead() uses CdReadyCallback() internally, so that function cannot be used with CdRead().

The return code from CdRead() only indicates if the command was issued successfully or not. For information about CD-ROM errors which occur during reading, check the result array of CdReadSync().

Return value

1 if command issued successfully, otherwise 0.

See also

[CdControl\(\)](#), [CdRead2\(\)](#), [CdReadSync\(\)](#), [CdReadCallback\(\)](#)

CdRead2

Start reading data from the CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdRead2(
int mode)
```

CD-ROM subsystem mode, as defined for CdISetmode command of CdControl()

Explanation

Seeks to the position specified by CdISetloc and starts reading data into the internal sector buffer. Starts streaming when the CdIModeStream flag is set in *mode* (see Table 10-4 for a list of modes). Starts ADPCM audio play when the CdIModeRT flag is set in the *mode* parameter. CdIModeSpeed can be used for multi-speed play.

This function must be used in conjunction with CdGetSector() to transfer data from the internal sector buffer to the program's desired destination buffer. CdGetSector() should be called to transfer data as soon as either CdReady() or CdReadyCallback() return the CdIDataReady flag.

Return value

1 if command issued successfully, otherwise 0.

See also

[CdControl\(\)](#), [CdRead\(\)](#), [CdGetSector\(\)](#), [CdReady\(\)](#), [CdReadyCallback\(\)](#)

CdReadBreak

Interrupt CdRead().

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	4.0	12/14/98

Syntax

void CdReadBreak(void)

Explanation

Used to interrupt CdRead(). Data which was read up until the interrupt is not secured.

Executing CdReadBreak() immediately after executing CdRead() (when 1 sector has not been read), can cause an error.

See also

[CdRead\(\)](#)

CdReadCallback

Define a callback function to be executed on completion of CdRead().

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

u_long CdReadCallback(*void(*func)*) Pointer to callback function address

void (*func)(*int status*, Return code of the CdReadSync()
u_char *result) Pointer to an 8-byte array containing status and result
 information

Explanation

func defines the function to be called when CdRead() completes. If *func* is 0, callback does not occur. *func* is passed two arguments:

- *status* is either CdlComplete or CdlDiskError, corresponding to the return code of CdReadSync().
- *result* is a pointer to an 8-byte array containing status and result information, corresponding to the *result* argument of CdReadSync().

While *func* is executing, subsequent data transfer complete interrupts are masked. Therefore, *func* should return as soon as the necessary processing is completed.

Return value

Address of previously set callback. Can be used to restore the previous callback when processing ends.

See also

[CdRead\(\)](#), [CdReadSync\(\)](#)

CdReadExec

Load PlayStation-format executable program file from CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

```
struct EXEC *CdReadExec(  
char *file)           Pointer to executable file name
```

Explanation

Loads the executable program specified by *file* into main memory at the address specified by the program file header. It is a blocking function.

After loading, the program can be executed as a child process using Exec(). The load address of the executable file should not overlap with the region of its parent process.

Return value

Pointer to an EXEC structure that describes the loaded program.

CdReadFile

Read a CD-ROM file.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

```
int CdReadFile(
  char *file,           Pointer to file name
  u_long *addr,         Pointer to main memory address to be read-in
  int nbyte)            Data size to be read-in
```

Explanation

Reads *nbyte* of the file on CD-ROM. *nbyte* must be a multiple of 2048; reads the entire file if *nbyte* is 0. *file* must contain a full path specification. All lowercase letters are converted to uppercase. When *file* is NULL, it starts reading from the next byte after the byte read by the last CdReadFile().

Although reading is performed in the background, because CdSearchFile() is called internally before reading begins, it is blocked for that period. Use CdReadSync() to determine when reading is completed.

Return value

Number of bytes read, if successful. On error, returns 0.

See also

[CdSearchFile\(\)](#), [CdReadSync\(\)](#)

CdReadSync

Check completion of CdRead() and related functions.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

int CdReadSync(

int *mode*, *Await read completion*

u_char **result*) *Pointer to status storage buffer of command most recently completed*

Explanation

Checks the current status of a data read operation initiated by CdRead(), CdReadFile(), and related functions. If *mode* is 0, the function waits for the operation to complete. If *mode* is 1, it returns the current status immediately.

Return value

Number of sectors remaining. If operation completed, 0 is returned. On error, -1 is returned.

See also

[CdRead\(\)](#), [CdReadSync\(\)](#)

CdReady

Wait for CD-ROM data to be ready.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdReady(
  int mode,           Wait until data is prepared
  u_char *result)    Pointer to status storage buffer of command most recently completed.
```

Explanation

Used after a CD-ROM read is initiated using CdRead2(), CdControl (CdIReadS), or CdControl (CdIReadN) to determine if there is data available in the sector buffer which is ready to be transferred using CdGetSector().

If *mode* is 0, the function waits for the operation to complete. If *mode* is 1, it returns the current status immediately.

Return value

Status can be one of the following:

CdIDataReady	There is data available for transfer
CdIDiskError	Error detected
CdINolntr	No preparation-completed data

See also

[CdReadyCallback\(\)](#), [CdRead2\(\)](#), [CdControl\(\)](#), [CdGetSector\(\)](#)

CdReadyCallback

Define CdReady callback function.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

u_long CdReadyCallback(*void(*func)*)

void (**func*)(**int** *status*,
u_char **result*)

Pointer to callback function address

Processing status of read command
Pointer to an 8-byte array containing status and result information

Explanation

Defines a callback routine *func* to be executed when data is available in the sector buffer following a CD-ROM read initiated using CdRead2(), CdControl (CdIReadS) or CdControl (CdIReadN). If *func* is NULL, any previous callback routine is disabled.

func is passed two arguments:

- status* is either CdIComplete or CdIDiskError, corresponding to the return code of CdSync() (although CdSync() is not called).
- result* is a pointer to an 8-byte array containing status and result information, corresponding to the *result* argument of CdSync().

While *func* is executing, subsequent data available interrupts are masked. Therefore *func* should return as soon as the necessary processing is completed.

Return value

Address of previously set callback. Can be used to restored the previous callback.

See also

[CdReady\(\)](#), [CdRead2\(\)](#), [CdControl\(\)](#), [CdSync\(\)](#)

CdReset

Initialize CD-ROM subsystem.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdReset(
int mode)           Reset mode
```

Explanation

Initializes the CD-ROM subsystem. Lower-level alternative to CdInit().

Unlike CdInit(), this function does not initialize the event environment related to CD-ROM.

mode can be:

- 0: Initialization of CD subsystem only (volume settings specified in previous sound libraries are saved)
- 1: Initialization of CD subsystem and CD audio volume (CD-DA, ADPCM)

No retry is carried out. Since CdInit() and CdReset() reset the SPU sound volume and CD input volume to the SPU, etc., they must be called before libspu/libsnd initialization and setting functions.

Return value

1 if initialization successful; 0 if unsuccessful.

See also

[CdInit\(\)](#)

CdSearchFile

Get location and size from CD-ROM file name.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
CdIFILE *CdSearchFile(
CdIFILE *fp,           Pointer to CD-ROM file structure pointer
char *name)            Pointer to a file name
```

Explanation

Determines the position time code (minutes, seconds, sectors) and total length of the specified file on the CD-ROM. The result is stored in the CdIFILE structure pointed to by *fp*.

name must be a complete path to the file.

CdSearchFile() caches directory information, so subsequent consecutive calls for files in the same directory do not require additional CD-ROM reads. Only one directory is cached at a time, and reading information for a file in another directory invalidates the entire cache.

For the best possible performance, include file location and size information in your program at compile time instead of using CdSearchFile().

Return value

Pointer to the CD-ROM file structure obtained; 0 if file not found.

CdSetDebug

Set debug level.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	2/24/99

Syntax

```
int CdSetDebug(
int level)           Debug level
```

Explanation

Set debug level for CD-ROM subsystem. The possible values of *level* are:

- 0: No checks performed
- 1: Check primitive commands
- 2: Print execution status of primitive commands
- 3: Print issued command error

Return value

Previously set debug mode

CdStatus

Get latest CD-ROM status.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.0	12/14/98

Syntax

int CdStatus(void)

Explanation

Obtains the latest reported CD-ROM status.

This function operates at high speed because it simply returns the status code maintained by the CD-ROM system. The status buffer is updated whenever a CD-ROM command is issued. To explicitly obtain the absolute most current status, issue a CdControl(CdINop) command immediately before your CdStatus() call.

Return value

CD-ROM Status.

CdSync

Wait for or check completion of CD-ROM command.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int CdSync(
int mode,           Waits for command termination
u_char *result)     Pointer to status storage buffer of command most recently completed.
```

Explanation

If *mode* is 0, waits for command termination and returns. If *mode* is 1, determines current status and promptly returns.

Return value

Command execution status is indicated by the following values:

CdIComplete:	Command complete
CdIDiskError:	Error detected
CdINolntr:	Command is being executed

See also

[CdSyncCallback\(\)](#)

CdSyncCallback

Define CdSync callback function.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

u_long CdSyncCallback(void(*func)) Pointer to callback function address

void (*func)(int status, Return code of CdSync()
u_char *result)) Pointer to an 8-byte array containing status and result information

Explanation

Defines a callback routine *func* to be executed when a CdControl() command is completed. If *func* is NULL, any previous callback routine is disabled.

func is passed two arguments:

- *status* is either CdlComplete or CdlDiskError, corresponding to the return code of CdSync().
- *result* is a pointer to an 8-byte array containing status and result information, corresponding to the *result* argument of CdSync().

While *func* is executing, subsequent CD-ROM command complete interrupts are masked. Therefore, *func* should return as soon as the necessary processing is completed.

Return value

Address of previously set callback. Can be used to restore previous callback.

See also

[CdSync\(\)](#)

StCdInterrupt

Handler for interrupts from CD-ROM (internal function).

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

void StCdInterrupt(void)

Explanation

Used as the CdReadyCallback routine by StSetStream() and StSetEmulate(). It transfers sectors from the CD controller to the streaming ring buffer as they become available. This function does not need to be called directly by the user when playing movies in 16-bit mode.

When playing a movie in 24-bit mode, there is a potential hardware conflict between the CD subsystem and the MDEC image decompression system which can result in corrupted data. To avoid this, StCdInterrupt() may defer transferring a sector and instead set a flag variable called StCdIntrFlag to indicate that a CD sector is ready to be transferred. Once the MDEC is finished transferring data, your application should check StCdIntrFlag and call StCdInterrupt() directly if it is set. Please consult the Sony sample code for movie playback for examples of the proper workaround.

See also

[CdGetSector\(\)](#), [DsGetSector\(\)](#)

StClearRing

Flush ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

void StClearRing(void)

Explanation

Flush ring buffer. Flushing the ring buffer when jumping tracks is effective in preventing excess frames from showing up.

StFreeRing

Release ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

**u_long StFreeRing(
u_long *base)** Pointer to starting address of user data area of released 1 frame

Explanation

The area obtained by StGetNext() is locked. StFreeRing() releases this locked region. The released region is the region for one frame's worth of data which is used as the base for the starting address of the user region. Linked sector header regions are also released.

If a region locked by StGetNext() is not released when its use ends, the ring buffer will overflow and streaming will come to a halt.

Return value

0 if release succeeded; 1 if release failed (for example, trying to release something that wasn't locked).

StGetBackloc

Return location and ID of first frame in the ring buffer in order to avoid any frame skip.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

int StGetBackloc(

CdILOC *loc)

Pointer to latest location of the first frame.
(use **DsILOC *loc** when using libds)

Explanation

Returns the latest location information and ID of the first frame in the current ring buffer.

The location information is used as the access target value in order to avoid frame skip due to ring buffer overflow. The frame skip due to ring buffer overflow can be avoided by re-accessing the frame location obtained by this function. This function is not appropriate for data with XA audio since it requires data access.

Please refer to `\psx\sample\cd\movie\tuto3.c` for usage example.

This function is valid only for CdlModeStream2 mode.

Return value

Frame ID that should be used on restart of streaming. -1 for error indicating non-StModeStream2 mode.

StGetNext

Get one frame of ring buffer data.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
u_long StGetNext(
u_long *addr,           Pointer to starting address of user data region for 1 frame of retrieved
                           data
u_long *header)        Pointer to starting address of sector header region for 1 frame of
                           retrieved data
```

Explanation

Gets one frame of ring buffer data. If the next frame of data is ready in the ring buffer, the starting address of the user data and the sector header are stored in *addr* and *header* respectively.

The region the data is taken from is locked until StFreeRing() is called, so it cannot be destroyed by new data.

The data region has a contiguous address and the ring buffer does not loop in mid-frame.

Return value

0, if a frame of data is taken from the ring buffer. If it is not ready, 1 is returned.

See also

[StGetNextS\(\)](#), [StFreeRing\(\)](#)

StGetNextS

Get one frame of ring buffer data from memory.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

u_long StGetNext(

u_long **addr*,

Pointer to user data region starting address for 1 frame of retrieved data

u_long **header*)

Pointer to sector header region starting address for 1 frame of retrieved data

Explanation

Gets one frame of ring buffer data. The starting addresses and the sector header are stored in *addr* and *header* respectively.

Return value

0, when one frame of data is taken from the ring buffer.

See also

[StGetNext\(\)](#)

StNextStatus

Return status of the next frame.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

```
u_long StNextStatus(
u_long *addr,           Pointer to starting address of the user data region for 1 frame of retrieved
                        data
u_long *header)       Pointer to starting address of sector header region for 1 frame of retrieved
                        data
```

Explanation

Obtains the status of the next frame of ring buffer data. The internal state is not affected by calling this function.

Return value

Status can be:

StFREE	Next frame is not in the ring buffer.
StCOMPLETE	Next frame is completely read into the ring buffer.
StBUSY	Next frame is being read into the ring buffer.
StLOCK	Next frame is being processed; i.e. one frame is obtained by calling StGetNext() but StFreeRing() has not been called.

StRingStatus

Return status of ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	3.5	12/14/98

Syntax

void StRingStatus(

short *free_sectors,

short *over_sectors)

Pointer to the number of free sectors on the ring buffer.

Pointer to the difference between the sector positions of CD-ROM data read in and the sector positions currently being processed.

Explanation

Reports the ring buffer status with two variables specified as arguments:

- *free_sectors* is the number of sectors with no data in the unused area of the ring buffer. The larger *free_sectors* is, the more free space in the ring buffer.
- *over_sectors* is the difference between the sector positions for CD-ROM data read in and the sector positions currently being processed. The larger *over_sectors* is, the more unprocessed data in the ring buffer.

The sum of *free_sectors* and *over_sectors* and the total ring buffer size is nearly equal. The reason for not having an exact match in size is that when one frame cannot fit in completely close to the end, rewind occurs.

Frame skip caused by insufficient free space in the ring buffer can be detected by calling this function.

StSetChannel

Set streaming channel.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
int StSetChannel(  
u_long ch)           Playback channel
```

Explanation

Sets streaming playback channel to *ch* (0-31). The channel stores the STR data at the authoring level.

Return value

0 if the channel is set; 1 otherwise.

StSetEmulate

Set parameters for streaming emulation.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	2/24/99

Syntax

```
void StSetEmulate(  
  u_long *addr,           Pointer to emulation data starting address  
  u_long loc,             Set color mode  
  u_long start_frame,     Streaming start frame  
  u_long end_frame,       Streaming end frame  
  void (*func1)(),        Address of function called back for each frame of data. If 0, no callback  
                           occurs.  
  void (*func2)())        Address of function called back when streaming ends. If 0, no callback  
                           occurs.
```

Explanation

Sets parameters for streaming emulation. Emulation means that CD-ROM data is put into memory in advance and data streaming is performed from memory, not from the CD-ROM, which provides only data-ready timing. In streaming emulation, play time is limited to a few seconds because of limits in memory capacity. Still, emulation is easier than using a CD-ROM emulator.

STR-format data needs to be loaded to *addr* in advance. See `StSetStream()` for details on other arguments. (*loc* is the same as *mode*.)

See also

[StSetStream\(\)](#)

StSetMask

Control the playing of streaming.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
void StSetMask(
u_long mask,           Streaming play on/off
u_long start,          StSetStream() start_frame
u_long end)            StSetStream() end_frame
```

Explanation

Turns streaming play ON/OFF. There is no mechanical timing lag compared to CD-ROM drive pause and playback, and instant ON/OFF is possible.

mask is 0 for Play, and 1 for Pause.

Resets start and end of `StSetStream()` trigger frame values.

See also

[StSetStream\(\)](#)

StSetRing

Set ring buffer.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
void StSetRing(  
u_long *ring_addr,      Pointer to ring buffer starting address  
u_long ring_size)      Ring buffer size (in sectors)
```

Explanation

Secures a ring buffer of a size specified by *ring_size* from an address specified by *ring_addr*. To use the Streaming Library, you must first call this function.

Because only form-1 CD-ROM sectors are supported at present, one sector of data area is 2048 bytes.

See also

[StUnSetRing\(\)](#)

StSetStream

Set streaming parameters.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

```
void StSetStream(
  u_long mode,           Set color mode
  u_long start_frame,    Frame to start streaming
  u_long end_frame,      Frame to end streaming
  void (*func1)(),       Address of function called back for each frame of data. If 0, no callback
                          occurs.
  void (*func2)(),       Address of function called back when streaming ends. If 0, no callback
                          occurs.
```

Explanation

Sets streaming parameters. Argument are as follows:

- *mode* sets color mode. 0 = 16-bit mode; 1 = 24-bit mode.
- *start_frame* specifies the frame number (stored in STR data) that starts streaming. Streaming doesn't begin until this frame is reached. If you want to play the data starting in the middle, you must specify an appropriate frame number. When you specify 0, streaming commences no matter what the frame number is.
- *end_frame* specifies the frame number (stored in STR data) that ends streaming. Streaming ends when this frame is reached. If you specify a number large enough, it plays the CD-ROM data to the end and terminates. When you specify 0, all the data is stored in the ring buffer and the function automatically terminates. This takes a large ring buffer, and the function is successful when streaming is from memory.
- *func1* is the address of the callback function called when one frame's worth of data is generated.
- *func2* is the address of the callback function called when streaming is completed.

To correctly exit from a streaming application, the end of streaming should not be set by *end_frame*. Set *end_frame* to 0xffffffff, and code an appropriate endpoint from within the loop.

StUnSetRing

Release interrupt used by streaming library.

Library	Header File	Introduced	Documentation Date
<i>libcd.lib</i>	<i>libcd.h</i>	2.x	12/14/98

Syntax

void StUnSetRing(void)

Explanation

Release two interrupt functions CdDataCallback() and CdReadyCallback() hooked by CDRead2(CdlModeStream) and return to initial state.

If the streaming library is not used when streaming ends and control transfers to another program, the interrupt hooks which call this function need to be returned to the initial state.

It is necessary to link libds when using this function.

See also

[StSetRing\(\)](#), [StSetStream\(\)](#)

Chapter 11: Extended CD-ROM Library

Table of Contents

Structures

DsIATV	11-4
DsIFILE	11-5
DsIFILTER	11-6
DsILOC	11-7

Functions

DsClose	11-8
DsCommand	11-9
DsComstr	11-10
DsControl	11-11
DsControlB	11-12
DsControlF	11-13
DsDataCallback	11-14
DsDataSync	11-15
DsEndReadySystem	11-16
DsFlush	11-17
DsGetDiskType	11-18
DsGetSector	11-19
DsGetSector2	11-20
DsGetToc	11-21
DsInit	11-22
DsInstr	11-23
DsIntToPos	11-24
DsLastCom	11-25
DsLastPos	11-26
DsMix	11-27
DsPacket	11-28
DsPlay	11-29
DsPosToInt	11-30
DsQueueLen	11-31
DsRead	11-32
DsRead2	11-33
DsReadBreak	11-34
DsReadCallback	11-35
DsReadExec	11-36
DsReadFile	11-37
DsReadSync	11-38
DsReady	11-39
DsReadyCallback	11-40
DsReadySystemMode	11-41
DsReset	11-42
DsSearchFile	11-43
DsSetDebug	11-44
DsShellOpen	11-45
DsStartReadySystem	11-46
DsStatus	11-47
DsSync	11-48
DsSyncCallback	11-49

Structures

DslATV

Audio attenuator.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_char val0;      CD (L) -> SPU (L) attenuation
    u_char val1;      CD (L) -> SPU (R) attenuation
    u_char val2;      CD (R) -> SPU (R) attenuation
    u_char val3;      CD (R) -> SPU (L) attenuation
} DslATV;
```

Explanation

Structure for setting the CD volume (CD-DA and CD-XA).

The values for *val0* - *val3* can range from 0 to 128. For standard stereo volume adjustments,

val0 is set to the L channel volume

val1 is set to 0

val2 is set to the R channel volume

val3 is set to 0

DsIFILE

9660 file descriptor.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	<i>4.0</i>	<i>12/14/98</i>

Structure

```
typedef struct {
    DsILOC pos;           File position
    u_long size;          File size (in bytes)
    char name[16];        Filename
} DsIFILE;
```

Explanation

Stores the position and size of a type 9660 CD-ROM.

DsIFILTER

ADPCM channel.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Structure

```
typedef struct {  
    u_char file;      File number  
    u_char chan;      Channel number  
    u_short pad;       Reserved for system use  
} DsIFILTER;
```

Explanation

Specifies the ADPCM sector channel to be played back.

DsILOC

CD-ROM location.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_char minute;    Minutes
    u_char second;    Seconds
    u_char sector;    Sectors
    u_char track;     Track number (currently unused)
} DsILOC;
```

Explanation

Specifies the CD-ROM position. Each element is specified using BCD

See also

Functions

DsClose

Close the libds system.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

void DsClose(*void*)

Explanation

Closes the libds system, resets the libds kernel state machine, and detaches the callback function which controls the libds kernel that has been forked in the system.

This function must be called whenever control is passed to a child process, LoadExec() is performed, or when CD control functions outside of libds are used. Call DsInit() to reopen libds.

See also

[DsInit\(\)](#)

DsCommand

Add primitive command to the command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsCommand(
  u_char com,           Command code
  u_char *param,        Pointer to argument for command (u_char[4])
  DsICB cbsync,         Pointer to callback function
  int count)            Number of retries (0: no retries, -1: unlimited retries)
```

Explanation

Adds a command to the queue to be performed in the background. If execution of the command fails, it is retried *count* times. An error is returned if the command failed to complete after it was retried.

Separate callback functions can be set for each command. The callback triggers when the command completes (or returns an error). The execution status of a command can be obtained with DsSync().

Return value

The command ID (>0) if the command issued successfully, otherwise 0.

See also

[DsSync\(\)](#)

DsComstr

Get the character string corresponding to each command code (for debugging)

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.3	12/14/98

Syntax

```
char *DsComstr(      Command completion code
u_char com)
```

Explanation

Gets the corresponding character string from the process status code (used for debugging). For example, DslNop returns “DslNop”, DslSetloc returns “DslSetLoc”, and so forth.

Return value

Pointer to start of character string.

See also

[DslInit\(\)](#)

DsControl

CdControl() compatibility function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsControl(
  u_char com,           Command code
  u_char *param,        Pointer to arguments for command (u_char[4])
  u_char *result)       Pointer to storage for the return value (u_char[8])
```

Explanation

Provides the same interface as CdControl(). Unlike CdControl(), however, the command is handled such that the function blocks until the end of the operation, even if the command itself is non-blocking.

Return value

1: Execution of command was successful. 0: Execution of command failed.

See also

CdControl() (see libcd), [DsControlB\(\)](#), [DsControlF\(\)](#)

DsControlB

CdControlB() compatibility function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	<i>4.0</i>	<i>12/14/98</i>

Syntax

```
int DsControlB(
u_char com,           Command code
u_char *param,        Arguments for command (u_char[4])
u_char *result)       Return value for command (u_char[8])
```

Explanation

Provides the same interface as CdControlB(). The actual timing differs somewhat since the command queue is used.

Return value

1: Execution of command was successful. 0: Execution of command failed

See also

CdControlB() (see libcd), [DsControl\(\)](#), [DsControlF\(\)](#)

DsControlF

CdControlF() compatibility function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsControlF(
u_char com,           Command code
u_char *param)        Pointer to arguments for command (u_char[4])
```

Explanation

Provides the same interface as CdControlF() except for a few differences such as timing.

Internally, the specified command is simply added to the command queue. Note that the command ID is provided in the return value.

CdControlF() waits for the previous command to complete execution before issuing the new command. DsControlF(), on the other hand, adds the new command to the queue if the previous command has not completed execution.

Return value

The command ID (>0) if the command was successfully added to the queue; 0 otherwise.

See also

CdControlF() (see libcd), [DsControl\(\)](#), [DsControlB\(\)](#)

DsDataCallback

Set exit callback for DsGetSector() and DsGetSector2().

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
void (*DsDataCallback(
void(*func)() ))
```

Pointer to callback function

Explanation

Defines *func* as the callback to be executed on completion of a read operation initiated by DsGetSector() or DsGetSector2(). No callback is generated when *func* is set to 0.

This callback is really only useful with DsSector2(), since the transfer of data is finished when DsGetSector() exits.

Return value

Pointer to previous callback.

See also

[DsGetSector\(\)](#), [DsGetSector2\(\)](#), [DsDataSync\(\)](#)

DsDataSync

Wait for completion of DsGetSector2.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int DsDataSync(

int mode)

0: Wait for end of transfer

1: Check current status and return immediately

Explanation

Waits for the transfer performed by DsGetSector2() to complete.

Return value

1: transfer is in progress. 0: transfer is complete

See also

[DsGetSector\(\)](#), [DsGetSector2\(\)](#), [DsDataCallback\(\)](#)

DsEndReadySystem

End simple callback system.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

void DsEndReadySystem(void)

Explanation

Ends simple callback system.

This function is executed within the callback function provided to DsStartReadySystem().

See also

[DsStartReadySystem\(\)](#)

DsFlush

Flush the command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

void DsFlush(void)

Explanation

All commands that have been entered in the command queue are flushed. Currently executing commands are allowed to complete, but the results are not saved and callbacks are not invoked.

If a command is executing when this function is called, it is allowed to complete. Subsequent commands are put into a new queue.

DsGetDiskType

Get CD type.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	2/24/99

Syntax

```
int DsGetDiskType(void)
```

Explanation

Gets the type of CD currently installed: either a PlayStation (black) or non-PlayStation disk.

This function blocks until the system status (the status obtained from DsSystemStatus()) changes to DslReady.

The debugging station recognizes ISO9660 CDs (including CD-Rs) as type DslCdromFormat.

This function does not operate properly on the DTL-H2000.

This function internally changes the operation mode. Since the operation mode is DslModeSize1(0x20) when the function is complete, the mode must be reset as necessary.

Return value

DslCdromFormat	PlayStation® disk
DslOtherFormat	Any other type of CD
DslStatNoDisk	CD is not installed
DslStatShellOpen	CD cover is open

DsGetSector

Transfer data from the sector buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsGetSector(
void *madr,          Pointer to destination area in main memory
int size)            Transfer size (long word)
```

Explanation

Data is transferred from the sector buffer to the storage area in main memory pointed to by *madr*.

This function blocks until the end of the transfer operation.

The sector size varies according to the mode.

The data from the sector buffer can be transferred to memory over a number of iterations. The sector data in the buffer must be transferred to memory before it is overwritten by data from the next sector.

The transfer is complete when the function returns.

Return value

Always returns 1.

See also

[DsDataCallback\(\)](#), [DsDataSync\(\)](#), [DsGetSector2\(\)](#)

DsGetSector2

Transfer data from the sector buffer to main memory (non-blocking).

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsGetSector2(
void *madr,          Pointer to destination area in main memory
int size)            Transfer size (long word)
```

Explanation

Data is transferred from the sector buffer to the storage area in main memory pointed to by *madr*.

The transfer is performed in cycle-stealing mode so interrupts may be received during the transfer.

Since DsGetSector2() is a non-blocking function that can return after the transfer starts, the completion of transfer must be determined using DsDataSync() or DsDataCallback().

Receiving interrupts and accessing memory from the CPU are possible even during transfers in cycle-stealing mode. However, other DMA's are blocked until the transfer is completed.

Data transfers in cycle-stealing mode are more time-consuming compared to those in blocking mode (the mode used by DsGetSector()).

Return value

Always returns 1.

See also

[DsDataCallback\(\)](#), [DsDataSync\(\)](#), [DsGetSector\(\)](#)

DsGetToc

TOC read.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.1	12/14/98

Syntax

```
int DsGetToc(  
DslLOC *loc)          Location table
```

Explanation

The starting position of each track on the CD-ROM is obtained.

The largest track number is 100.

Return value

Positive integer: track number; Other values: error

Dslnit

Perform system initialization.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int Dslnit(void)

Explanation

Initializes the libds system.

Dslnit() needs to be called just once at the beginning of a program or when restarting a system that was stopped with DsClose().

Calling Dslnit() in the middle of a program may cause improper operation. DsReset() should be used if initialization needs to be performed in the middle of a program.

Because Dslnit() resets the SPU sound volume and the CD input volume to SPU, etc., it should either be called before libspu and libsnd initialization/setting functions or they should be reset after Dslnit() is called.

Return value

1 if successful; 0 if the operation failed.

See also

[DsClose\(\)](#), [DsReset\(\)](#)

DsInstr

Get the corresponding character string for the command process status (for debugging).

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.3	12/14/98

Syntax

```
char * DsInstr(           Execution status code
u_char intr)
```

Explanation

For debugging. Gets the corresponding character string from the process status code.

Table 11-1

Process status	Character string
DsINolntr	"Nolntr"
DsIComplete	"Complete"
DsIDiskError	"Disk Error"

Return value

Pointer to start of character string.

See also

[DsComstr\(\)](#), [DsSetDebug\(\)](#)

DsIntToPos

Get minutes, seconds, sectors from absolute sector number.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

[DsLOC](#) *DsIntToPos(

[int](#) *i*, Absolute sector number
[DsLOC](#) **p*) Pointer to buffer for storing result

Explanation

The absolute sector number specified by *i* is converted to minutes, seconds, and sectors and the result is stored in the DsLOC structure pointed to by *p*.

Return value

Pointer to result buffer

See also

[DsPosToInt\(\)](#)

DsLastCom

Get the command issued last.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.3	9/1/99

Syntax

u_char DsLastCom(void)

Explanation

Returns the primitive command code issued last.

However, because this function is replaced by DslNop in libds in the following situations, the desired value may not be returned.

- When DslPause is issued and then reissued it after it is successful.
- When DslStop is issued while the spindle is stopped.
- When DslStandby is issued while the spindle is revolving.

Return value

Primitive command code

See also

[DsControl\(\)](#)

DsLastPos

Get the last setloc position.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

`DsLOC *DsLastPos(
DsLOC *p)` Pointer to buffer in which position is stored

Explanation

The last setloc position is obtained and the result is stored in the DsLOC structure pointed to by *p*.

Return value

Pointer to result buffer

DsMix

Set attenuator.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	<i>4.0</i>	<i>12/14/98</i>

Syntax

```
int DsMix(  
    DslATV *vol)    Attenuator volume
```

Explanation

The CD audio volume (CD-DA/ADPCM) is set to the value in the DslATV structure pointed to by *vol*.

Return value

1.

DsPacket

Add a sequence of commands to the queue.

Library	Header File	Introduced	Documentation Date
libds.lib	libds.h	4.0	12/14/98

Syntax

int DsPacket(
u_char mode, Operating mode
DsLOC *pos, Pointer to DsLOC structure specifying target position
u_char com, Last command to be executed
DsICB *cbsync, Callback function to be triggered when all the commands have been executed
int count) Retry count (0: no retries, -1: unlimited retries)

Explanation

Adds a sequence of commands that perform a data read (playback) to the queue.

The commands added to the queue are: DsIPause; DsISetmode *mode*; DsISetloc *pos*; *com*.

The commands that can be specified for *com* are: DsIPlay, DsIReadN, DsIReadS, DsISeekP, or DsISeekL. If *com* is DsIPlay, DsIReadN, or DsIReadS, execution is performed up to and including the data read (playback). If *com* is DsISeekP or DsISeekL, the seek is performed and the system enters a pause state.

If any command in the sequence generates an error, a retry is performed starting with the first command, up to a maximum of *count* times (or unlimited times if *count*=-1). An error is generated if the operation is not successful after *count* retries.

DsSync() can be used to obtain the execution status. When all the commands in the sequence are successful or if an error is generated, the callback function *cbsync* is triggered.

An error is generated if the queue does not have enough space for the command sequence.

Return value

The command ID (>0) if the command was added to the queue; 0 if the command failed.

See also

DsCommand(), DsQueueLen()

DsPlay

Play back CD-DA tracks.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int DsPlay(

int mode,

int *tracks,

int offset)

Mode

Pointer to array specifying the tracks to be played back; the last element of the array must be 0.

Index for track to begin playback

Explanation

The tracks specified by the *tracks* array are played in sequence in the background.

When the final track in the series is done, playback is repeated or is stopped, depending on *mode*. The values available for *mode* are shown below.

Table 11-2

Mode	Description
0	Stop playback
1	Play back the tracks in sequence; then stop playback.
2	Play back the tracks in sequence; then repeat from beginning.
3	Return the index of the track currently being played

Playback is performed in increments of tracks. Playback cannot start or stop in the middle of a track.

Return value

The track currently being played (the index in the *tracks* array rather than the absolute track number).

-1 means that all tracks have finished playing.

DsPosToInt

Get absolute sector number from minutes, seconds, sectors.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int DsPosToInt(
[DsLOC *p](#)) Pointer to DsLoc structure containing minutes, seconds, sectors

Explanation

Calculates the absolute sector number from the minutes, seconds, and sectors in the DsLOC structure pointed to by *p*.

Return value

The absolute sector number.

See also

[DsIntToPos\(\)](#)

DsQueueLen

Get the number of commands stored in the command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsQueueLen(void)
```

Explanation

Obtains the number of primitive commands stored in the command queue.

The commands issued by `DsPacket()` are not removed from the queue until they all successfully complete. Therefore the number of commands returned by `DsQueueLen()` remains unchanged during execution of the packet.

The currently executing command is considered to be in the queue. The maximum number of commands that the queue can hold is defined by the `DsMaxCOMMANDS` macro constant.

Return value

Number of commands in queue.

See also

[DsPacket\(\)](#)

DsRead

Read data.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsRead(
    DsILOC *pos,           Pointer to DsILOC structure specifying starting position of CD
    int sectors,           Number of sectors to read
    u_long *buf,           Pointer to buffer to store read data
    int mod)               Operating mode to be used when data is being read
```

Explanation

Reads CD data starting at the location specified by the DsILOC structure pointed to by *p*. The data is stored in the buffer pointed to by *buf*.

The operation is performed in sectors, so the size of *buf* must be a multiple of 1 sector=2048 bytes (512 words).

Reading is performed in the background after DsRead() has executed and exited. Successful execution of the function does not indicate that the data has been successfully read.

Note: The arguments are different from CdRead(). With DsRead(), the starting position of the data must be specified.

Return value

Positive integer: the id of the packet that was issued within the function, if execution was successful.

0: function execution failed.

See also

[DsReadBreak\(\)](#), [DsReadCallback\(\)](#), [DsReadSync\(\)](#), [CdRead\(\)](#) (see libcd)

DsRead2

Begin playback of movie.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsRead2(
    DsILOC *pos,           Pointer to DsILOC structure specifying starting position of CD
    int mod)               Operating mode during playback
```

Explanation

Plays back the movie starting at the location specified by the DsILOC structure pointed to by *pos*.

A libds streaming library callback is set and the reading of data is begun with DsIReadS.

Note: The arguments are different from CdRead2(). With DsRead2(), the starting position of the data must be specified.

Return value

The command ID (>0) if the function succeeded; 0 if the command failed.

See also

[DsCommand\(\)](#), [CdRead2\(\)](#) (see [libcd](#))

DsReadBreak

Interrupt DsRead() operation.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

void DsReadBreak(void)

Explanation

Interrupts a DsRead() operation.

See also

[DsRead\(\)](#)

DsReadCallback

Set a callback function to be called when DsRead() is finished.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

DsICB DsReadCallback(
DsICB *func*) Pointer to callback function

Explanation

Defines *func* as the callback to be triggered when DsRead() completes.

Return value

Pointer to previous callback function

See also

[DsRead\(\)](#)

DsReadExec

Read an executable file.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
struct EXEC *DsReadExec(
char *file)           Filename
```

Explanation

Loads the executable file specified by *file* from the CD-ROM and stores it in main memory. It is a blocking function.

The loaded file is executed as a child process using Exec(). The load address of the executable file must not overlap with the area used by the parent process.

Return value

Pointer to EXEC structure of the loaded executable file.

See also

[Exec\(\)](#) (see libapi)

DsReadFile

Read a file from CD-ROM.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsReadFile(
char *file,           Filename
u_long *addr,        Pointer to buffer in memory for storing read data
int nbyte)           Number of bytes to read
```

Explanation

Reads *nbyte* bytes from the CD-ROM file specified by *file* and stores them at the buffer pointed to by *addr*.

nbyte must be a multiple of 2048; if it is 0, the entire file is read. If *file* is NULL, the read operation begins from the point where the previous DsReadFile() left off.

The filenames must all be represented by absolute paths. Lowercase characters are automatically converted to uppercase.

Although the read is performed in the background, DsSearchFile() is called internally before the read begins, so it is blocked for that period. Use DsReadSync() to check for completion of reading.

Return value

The number of bytes read, or 0 if an error occurred.

See also

[DsRead\(\)](#), [DsReadSync\(\)](#), [DsSearchFile\(\)](#)

DsReadSync

Wait for completion of DsRead().

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsReadSync(  
u_char *result)          Pointer to buffer holding execution results (u_char[8])
```

Explanation

Waits for completion of DsRead(). Returns the execution status of DsRead() at the point when DsReadSync() was called.

Return value

Positive integer: the remaining number of sectors.
0: DsRead() has completed.
-1: an error was detected (DsRead() was interrupted).

See also

[DsRead\(\)](#)

DsReady

Check for arrival of data.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsReady(
u_char *result)           Pointer to buffer for storing results (u_char[8])
```

Explanation

Determines the status of a data read operation (DsIReadS/DsIReadN) and stores the result in the buffer pointed to by *result*.

In report mode, DsReady() checks for arrival of the report from DA playback.

The sector buffer value is meaningful only for data reads.

Return value

DsIDataReady	New data has arrived in the sector buffer.
DsINoIntr	New data has not arrived.
DsIDataEnd	Final sector has been confirmed (only for DA playback).

See also

[DsReadyCallback\(\)](#)

DsReadyCallback

Set up Ready callback function.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

DsICB DsReadyCallback(
DsICB *func*) Pointer to callback function

Explanation

Sets the Ready callback to the function pointed to by *func*. It is called for data ready interrupts, data end interrupts (generated only for DA playback), and all error interrupts.

Return value

Pointer to previous callback function.

See also

[DsReady\(\)](#)

DsReadySystemMode

Set action of cover open/close for the simple callback.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.1	12/14/98

Syntax

```
int DsReadySystemMode(
int mode)
```

0: When the cover is open, end the simple callback

1: After the cover opens or closes, perform an automatic retry

Explanation

Sets the action of cover open/close for the simple callback.

When *mode* = 0, if the cover is opened during execution, stop processing and end the simple callback. Then call the user-specified callback function with *intr* = *DslDiskError*.

When *mode* = 1, if the cover is opened or closed, reissue the command with an error and continue processing. The user-specified callback function is not called.

When *mode* = 1 and the cover is closed, if the disk is not present, the simple callback is completed and the user-specified callback function is called with *intr* = *DslDiskError*.

Initial value of *mode* is 0.

The mode is valid until the next time it is set.

Return value

Previously updated mode.

See also

[DsStartReadySystem\(\)](#), [DsEndReadySystem\(\)](#)

DsReset

Reset system.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int DsReset(void)

Explanation

Resets the libds system.

Always use DsReset() when initializing the system in the middle of a program. DsInit() cannot be used in the middle of a program.

Return value

1 if the reset was successful, 0 otherwise.

See also

[DsInit\(\)](#)

DsSearchFile

Get position and size of CD-ROM file.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
DsIFILE *DsSearchFile(
DsIFILE *fp,           Pointer to CD-ROM file structure
char *name)           Filename
```

Explanation

Obtains the absolute position (minutes, seconds, sectors) and size of the CD-ROM file specified by *name* and stores the result in the DsIFILE structure pointed to by *fp*.

Filenames must be represented by their absolute paths.

The position data for all the files in the same directory as the file specified by *fp* is cached in memory. Therefore, when DsSearchFile() is performed consecutively for files from a single directory, access is faster from the second file on.

Return value

0: file not found.
 -1: the read operation on the directory failed for some reason.
 Other: pointer to the retrieved file structure.

DsSetDebug

Set the debug level.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsSetDebug(  
int level)           Debug level
```

Explanation

Sets the debug level for the CD-ROM subsystem to *level*:

0: Do not perform any checks

1: Check primitive commands

Return value

Previous debug level.

DsShellOpen

Get the number of times the cover was opened.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int DsShellOpen(void)

Explanation

Obtains the number of times the cover was opened since the program began. The count is initialized to 1 when the program starts.

Note: This function returns the correct value only when DsSystemStatus()=DsReady.

Return value

Number of times the cover was opened.

See also

[DsSystemStatus\(\)](#)

DsStartReadySystem

Start the simple callback.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsStartReadySystem(
    DsIRCB func,           Pointer to callback function
    int count)              Retry count (-1: unlimited retries)
```

Explanation

Starts the simple callback.

When the simple callback is started, a DsDiskError results in a retry of the last command.

count is the total number of retries from the point when the system is started.

The callback function *func* normally triggers when a data read successfully completes. The only time an error makes the function trigger is if the cover is opened or an error is generated after the maximum number of retries.

When a retry is performed, the position from which to re-read is determined by the library, but the callback function triggers from the sector following the previous call. Thus, internally, the callback function does not need to be aware of the retry.

This function is always executed from a callback from a corresponding data read (playback) command. Executing the function at other times may corrupt the error handling system.

DsReadyCallback() should be used internally for simple callback. Simultaneous use from the application is not allowed.

Return value

1: the function was successful.

0: the function failed (system has already been started).

See also

[DsEndReadySystem\(\)](#), [DsReadyCallback\(\)](#)

DsStatus

Get the status of the CD subsystem.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

u_char DsStatus(void)

Explanation

Obtains the last reported status of the CD subsystem.

Because updating of the status can sometimes be delayed, in rare cases the value may be different from the current CD subsystem status. To wait for any delays to pass, use DsINop to get the most recent status.

Return value

Status of the CD subsystem.

See also

[DsCommand\(\)](#)

DsSync

Check for completion of primitive command.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

```
int DsSync(
int id,           Command ID
u_char *result)  Pointer to buffer for storing result (u_char[8])
```

Explanation

Obtains the execution status of the primitive command specified by *id* and stores it in the memory area pointed to by *result*.

The execution status refers to the command corresponding to the command ID that was active when the function was called. *result* is valid only for the return values of DslComplete or DslDiskError.

If *id* is set to 0, the most current result regardless of the type of command can be obtained.

A certain number of execution results from commands are saved. The maximum number of saved results is defined by the DslMaxRESULTS macro constant.

Return value

DslComplete	Command has terminated normally.
DslDiskError	Command failed.
DslNoIntr	Command has not yet been executed.
DslNoResult	Execution has terminated but the results have already been destroyed.

See also

[DsSyncCallback\(\)](#)

DsSyncCallback

Set Sync Callback function

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

DsICB DsSyncCallback(
DsICB *func*) Pointer to callback function

Explanation

Defines *func* as the Sync callback to be triggered for all command termination and error interrupts.

If the queue performs retries for commands that generate errors, the Sync callback function is triggered after each failure (rather than the individual callback function set for the command itself).

Return value

Pointer to previous callback function.

See also

[DsSync\(\)](#)

DsSystemStatus

Get status of command queue.

Library	Header File	Introduced	Documentation Date
<i>libds.lib</i>	<i>libds.h</i>	4.0	12/14/98

Syntax

int DsSystemStatus(void)

Explanation

Returns the status of the command queue.

Commands issued when the status is not DslReady are all added to the queue; otherwise, the command is executed immediately.

Return value

DslReady	No command is being executed.
DslBusy	Command is being executed, or command cannot be executed (e.g., because cover is open).
DslNoCD	No CD is installed.

See also

[DsCommand\(\)](#)

Chapter 12: Controller/Peripherals Library

Table of Contents

Functions

CheckCallback	12-3
DisableTAP	12-4
EnableTAP	12-5
GetVideoMode	12-6
InitGUN	12-7
InitTAP	12-8
PadChkVsync	12-9
PadEnableCom	12-10
PadEnableGun	12-11
PadGetState	12-12
PadInfoAct	12-13
PadInfoComb	12-15
PadInfoMode	12-16
PadInit	12-18
PadInitDirect	12-19
PadInitGun	12-20
PadInitMtap	12-22
PadRead	12-23
PadRemoveGun	12-24
PadSetAct	12-25
PadSetActAlign	12-27
PadSetMainMode	12-28
PadStartCom	12-29
PadStop	12-30
PadStopCom	12-31
RemoveGUN	12-32
ResetCallback	12-33
RestartCallback	12-34
SelectGUN	12-35
SetVideoMode	12-36
StartGUN	12-37
StartTAP	12-38
StopCallback	12-39
StopGUN	12-40
StopTAP	12-41

CheckCallback

Determine whether the program is executing a callback.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	12/14/98

Syntax

int CheckCallback(void)

Explanation

Determines whether the program is currently executing in callback context or normal context.

Return value

0: normal context; 1: callback context.

See also

[ResetCallback\(\)](#)

DisableTAP

Disable communication with the controller

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.6	12/14/98

Syntax

void DisableTAP(void)

Explanation

Temporarily disables communication with the controller.

Although StopTAP() deletes the controller handler activated by Vsync interrupts, this function simply skips controller communication with a flag operation.

Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec when the controller status is not needed.

See also

[EnableTAP\(\)](#), [StopTAP\(\)](#)

EnableTAP

Enables occurrence of an event.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.6	12/14/98

Syntax

void EnableTAP(*void*)

Explanation

Enables communication with a controller which was disabled with DisableTAP().

Although a normal controller communicates via Vsync interrupts, this function is used only with timing longer than 1/60 sec when the controller status is not needed.

See also

[DisableTAP\(\)](#)

GetVideoMode

Get present video signaling system.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.1	12/14/98

Syntax

long GetVideoMode(void)

Explanation

Returns the present video signaling system set by SetVideoMode(). (If SetVideoMode() wasn't called, no matter what the machine, it returns MODE_NTSC.)

Return value

Video signaling system mode (MODE_NTSC for NTSC; MODE_PAL for PAL).

See also

[SetVideoMode\(\)](#)

InitGUN

Initialize gun.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.5	12/14/98

Syntax

```
void InitGUN(
char *bufA,           Controller receive data buffer for port 0
long lenA,            Length in bytes of bufA
char *bufB,           Controller receive data buffer for port 1
long lenB,            Length in bytes of bufB
char *buf0, char *buf1, Pointer to horizontal/vertical position receive buffer (necessary buffer size is
                        len*4+2 bytes)
long len)             Number of gun interrupts allowed between vertical blank periods (20
                        maximum)
```

Explanation

Defines the buffers used to receive data from the light gun and other controllers. Standard controller information for buttons and analog controllers is returned in *bufA* and *bufB*. InitGUN() cannot be used at the same time as InitPAD() or InitTAP().

As of library v4.0, DMA operations and interrupts are blocked within the gun interrupt handler in order to improve the accuracy of the gun.

The more gun interrupts you specify between VBLANK periods (*len*), the more processing is required. Set *len* as low as possible to reduce overhead.

Since the horizontal direction counter value returns the system clock value, multiply the following coefficients according to the horizontal direction resolution in order to obtain pixel values:

Table 12-1: System Clock/Pixel Clock Variable Table

Mode	Horizontal Direction Resolution	Coefficient
NTSC:	256	0.158532
	320	0.198166
	384	0.226475
	512	0.317065
	640	0.396332
PAL:	256	0.157086
	320	0.196358
	384	0.224409
	512	0.314173
	640	0.392717

[Pixel value] = [Coefficient] x [System Block value] + [Offset]

See also

InitPAD() (see libapi), [SelectGUN\(\)](#), [StartGUN\(\)](#), [StopGUN\(\)](#), [RemoveGUN\(\)](#)

InitTAP

Initialize controller.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.4	12/14/98

Syntax

void InitTAP(

char *bufA, Pointer to receive data buffer

long lenA, Receive data buffer length (unit: byte)

char *bufB, Pointer to receive data buffer

long lenB) Receive data buffer length (unit: byte)

Explanation

Registers a receive data buffer for the controller.

For the format of the receive buffer, see “Receive Buffer Data Format” of Chapter 13 (Controller/Peripherals Library) of the Library Overview.

Please refer to each terminal’s documentation for the physical positioning of the buttons and channels, etc. and compatibility.

See also

[StartTAP\(\)](#), [StopTAP\(\)](#)

PadChkVsync

Check communication with controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

int PadChkVsync(void)

Explanation

Determines whether communication with the controller has occurred in a frame. Should be called once per frame (1/60 sec) during Vsync.

Return value

- 1: Communication with controller took place (regardless of success/failure)
- 0: Communication with controller did not take place (or function was called twice or more in a frame)

See also

[PadEnableCom\(\)](#)

PadEnableCom

Enable/disable communication with the controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	2/24/99

Syntax

**unsigned PadEnableCom(
unsigned mode)**

Bit 0 is used to enable/disable port 0, and bit 1 is used to enable/disable port 1. 1 = enabled; 0 = disabled

Explanation

In general, communication with the controller takes place once per frame (1/60th of a second). However, when a lower update rate is desired (e.g. when polling for a button press), communication with the controller can be temporarily disabled with this function to provide the application with greater processing time.

The vertical retrace interrupt itself is not enabled or disabled, so PadEnableCom() only works between PadStartCom() and PadStopCom().

Ports 0 and 1 have a default value of "enabled." Calling PadInitDirect(), PadInitMtap(), PadStartCom(), or PadStopCom() don't affect the enable/disable state set by PadEnableCom().

If communication is suspended for three seconds or more, the controller is reset. If communication is subsequently restarted, the return value from PadGetState() temporarily becomes PadStateDiscon and a retry is generated to refetch controller information. For this reason, the return value from PadGetState() needs to be monitored so that refetched actuator information can be properly processed.

Inhibit communication with one port using PadEnableCom(). When the mouse is attached to the inhibited port, the controller which was attached to the enabled port will not recognize the invalid state of the mouse. Therefore, the same setting of enabling/inhibiting communication should be performed for both ports.

Return value

The previous enable/disable state of communication before the function was called.

See also

[PadStartCom\(\)](#), [PadStopCom\(\)](#)

PadEnableGun

Enable/disable gun interrupts.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

**void PadEnableGun(
u_char *mask*)** Enable gun interrupts for specific ports (see Explanation)

Explanation

Enables gun interrupts when the corresponding *mask* bit is set to 1:

Table 12-2

<i>mask</i> bits	D7	D6	D5	D4	D3	D2	D1	D0
Port number	0x13	0x12	0x11	0x10	0x03	0x02	0x01	0x00

Specific gun interrupts can be masked off if horizontal and vertical position information is not needed for those guns.

The default setting is *mask* disabled for all ports. Retrieval of horizontal and vertical position information begins when a gun is connected.

See also

[PadInitGun\(\)](#), [PadRemoveGun\(\)](#)

PadGetState

Get controller connection state.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	2/24/99

Syntax

int PadGetState(

int *port*)

The port number to be checked (see Explanation)

Explanation

Checks that the controller is connected, determines when button-press information is valid, and determines when information from the actuators is valid.

port represents the port number to be checked, as follows:

Table 12-3

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

Return value

Table 12-4

Value	Macro (libpad.h)	Controller connection state
0	PadStateDiscon	Controller disconnected
1	PadStateFindPad	Find controller connection (checking)
2	PadStateFindCTP1	Check for connection with a controller that does not support the vibration function (including SOPH-1150). (Complete the acquisition of controller information)
4	PadStateReqInfo	Actuator information being retrieved (data being retrieved)
5	PadStateExecCmd	Library is communicating with controller (e.g. PadSetActAlign())
6	PadStateStable	Check for connection with a controller that has a vibration function (DUAL SHOCK). Retrieval of actuator information completed, or library-controller communication completed (PadSetActAlign(), etc. can be called)

See also

[PadSetActAlign\(\)](#)

PadInfoAct

Get actuator information.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	2/24/99

Syntax

int PadInfoAct(

int *port*,

int *actno*,

int *term*)

Port number of the controller to be checked (see Explanation)

Actuator number to be checked (ranging from 0 to total number of mounted actuators -1). Set *actno* to -1 to get the total number of actuators (in this case, the third argument *term* is ignored)

Information to be checked about actuator

Explanation

Obtains the actuator function number, sub-function number, actuator parameter data size, and actuator current drain.

port is the port number of the controller to be checked, as follows:

Table 12-5

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

Return value

The return value corresponds to the third argument *term* as follows.

Table 12-6

Third argument	Macro	Return value
1	InfoActFunc	Function number (1: continuous-rotation vibration)
2	InfoActSub	Sub-function number (When the function number is 1, 1: low-speed rotation, 2: high-speed rotation)
3	InfoActSize	Parameter data length (0: 1 bit (ON/OFF only), 1 or greater: number of bytes)
4	InfoActCurr	Maximum current drain

If the parameter data length for an actuator is more than one byte, each of the parameter write offsets for that actuator can be set by writing the actuator number in each of the corresponding offsets using `PadSetActAlign()`. The controller interprets the position of the lowest numbered offset as the high-order byte of the parameter. If the actuator number is written such that the data length for the parameter is exceeded, the settings beyond the allowed parameter data length are ignored, beginning with the lowest numbered offset.

Note: Up to 60 units of current can be supplied by the main PlayStation® unit. Therefore, the current drain of the actuators should be checked to make sure that it does not exceed 60 units. If actuator parameters are set so that the 60-unit limit is exceeded, the actuators connected to the larger port numbers are ignored (they are forcibly stopped). This is particularly important for applications that use Multi Taps.

`PadInfoAct()` always returns a 0 for a controller that does not support the vibration function (including SCPH-1150). Even for a controller that has a vibration function (DUAL SHOCK), the return value is 0 as long as the state returned from `PadGetState()` is anything other than `PadStateStable` (during

12-14 Controller/Peripherals Library Functions

PadStateReqInfo). When obtaining actuator information, the application should wait for the return value from PadGetState() to become PadStateStable.

See also

[PadInfoComb\(\)](#), [PadInfoMode\(\)](#)

PadInfoComb

Get information on actuator combinations that can be used simultaneously.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	2/24/99

Syntax

int PadInfoComb(

int *port*,

int *listno*,

int *offs*)

Port number of the controller to be checked (see Explanation)

List number of the combination list to be checked (ranging from 0 to total combination list -1). Specify -1 for *listno* to obtain the total number of combination lists. In this case, the *offs* argument is ignored

Offset within the combination list (ranging from 0 to total number of actuators contained in the list -1). Specify -1 for *offs* to obtain the total number of actuators contained in the combination list.

Explanation

Checks combinations of actuators that can be used simultaneously based on restrictions imposed by the physical arrangement of the actuators, etc.

port is the port number of the controller to be checked, as follows:

Table 12-7

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

Return value

Table 12-8

<i>listno</i>	<i>offs</i>	Return value
-1	---	Total number of combination lists
0 to (Total number-1)	-1	Total number of actuators contained in list number <i>listno</i>
0 to (Total number-1)	0 to (Total number-1)	Actuator number stored at offset <i>offs</i> within list number <i>listno</i> .

Note: PadInfoComb() always returns 0 for a controller that does not support the vibration function (including SCPH-1150). Even for a controller that has a vibration function (DUAL SHOCK), the return value is 0 as long as the state returned from PadGetState() is anything other than PadStateStable (during PadStateReqInfo). When obtaining information about combinations of actuators that can be used simultaneously, the application should wait for the return value from PadGetState() to become PadStateStable.

See also

[PadInfoAct\(\)](#), [PadInfoMode\(\)](#)

PadInfoMode

Get information about the controller mode.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	2/24/99

Syntax

int PadInfoMode(

int port,

int term,

int offs)

Port number of the controller to be checked (see Explanation)

Item to be checked

The offset in the controller mode ID table containing the desired controller mode ID

Explanation

Checks the currently active controller mode ID, distinguishes DUAL SHOCK controllers from other controllers, and checks the controller mode ID supported by the DUAL SHOCK controller.

port is the port number of the controller to be checked, as follows:

Table 12-9

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

When DUAL SHOCK controller SCPH-1200 is connected and PadLoadInfo() is called, initialization completes and the return value is as shown below.

Table 12-10

Controller information	Contents immediately after initialization
Currently active controller mode ID	4
Currently active controller mode ID (Only for controllers with a vibration function)	4
Controller mode ID table offset for currently active controller mode ID	0
Contents of controller mode ID table	See table below

Table 12-11: Contents of controller mode table ID

Controller mode ID table offset	Controller mode ID
0	4
1	7

Return value

The return value corresponds to the second and third arguments(*) in the following manner.

Table 12-12

2 nd Argument	Macro	Return value
1	InfoModeCurlID	Currently active controller mode ID. Valid range: 4 bits. (Same as value of terminal type for offset 1 in receive buffer)
2	InfoModeCurExID	Currently active controller mode ID on a controller that has a vibration function. Valid range: 16 bits. (0 for controllers that do not support the vibration function)
3	InfoModeCurExOffs	Offset within the controller mode ID table that stores the currently active controller mode ID. (0 for controllers that do not support the vibration function.)
4	InfoModeldTable	Controller mode ID stored at the offset specified by the third argument <i>offs</i> within the controller mode ID table. (0 for controllers that do not support the vibration function.)

(*): If the second argument has a value other than 4(InfoModeldTable) the third argument *offs* is ignored.

When the second argument is 1 (InfoModeCurlID) or 2 (InfoModeCurExID), the function may be called at any time, regardless of the value returned by PadGetState().

When the second argument is 4 (InfoModeldTable), the return value will be 0 if PadGetState() does not return PadStateStable (for PadStateReqInfo). This is true even if the controller has a vibration function (DUAL SHOCK). After calling PadLoadInfo(), the application should wait for the return value from PadGetState() to become PadStateStable.

See also

[PadInfoAct\(\)](#), [PadInfoComb\(\)](#)

PadInit

Initialize a controller (for prototyping only).

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	12/14/98

Syntax

void PadInit(*mode*) Always pass 0

Explanation

Initializes the controller. Since this function supports only the 16-button controller, it should be used for prototyping purposes only.

See also

[PadInitDirect\(\)](#), [PadStop\(\)](#)

PadInitDirect

Initialize controller environment (for direct connection to the main PlayStation unit).

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

```
void PadInitDirect(
u_char *pad1,      Port 1 receive results (34 bytes)
u_char *pad2)      Port 2 receive results (34 bytes)
```

Explanation

Initializes the control environment for a controller.

When using this function, other initialization routines such as PadInitMtap(), InitPAD(), InitGUN(), InitTAP(), and PadInit() cannot be used.

In libpad, controller connection state is maintained by the library. If the connection state is invalid, the controller isn't be recognized. Therefore, when a controller is used by both parent and child processes, each process must call PadInitDirect().

For the format of the receive buffer, see "Receive Buffer Data Format" of Chapter 13 (Controller/Peripherals Library) of the Library Overview.

If a Multi Tap is not used, using this function for initialization reduces program size by about 1.6KB.

Meaning of analog values:

Table 12-13

Device	Analog value 1	Analog value 2	Analog value 3	Analog value 4
Analog controller	Position along the X axis (right) (0 ~ 80 ~ FF)	Position along the Y axis (right) (0 ~ 80 ~ FF)	Position along the X axis (left) (0 ~ 80 ~ FF)	Position along the Y axis (left) (0 ~ 80 ~ FF)
Analog joystick	Position along the X axis (right) (0 ~ 80 ~ FF)	Position along the Y axis (right) (0 ~ 80 ~ FF)	Position along the X axis (left) (0 ~ 80 ~ FF)	Position along the Y axis (left) (0 ~ 80 ~ FF)
Gun controller (Namco)	Position along the X axis	Position along the X axis	Position along the Y axis	Position along the Y axis
Mouse	Low-order byte Displacement along the X axis (80 ~ 0 ~ 7F)	High-order byte Displacement along the Y axis (80 ~ 0 ~ 7F)	Low-order byte	High-order byte
			None	None

See also

[PadInit\(\)](#), [PadInitMtap\(\)](#), [PadInitGun\(\)](#), [PadStartCom\(\)](#), [PadStopCom\(\)](#)

PadInitGun

Initialize controller environment (for guns that use interrupts).

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

```
void PadInitGun(
u_char *buff,           Horizontal/vertical position receive buffer (required buffer size = size*4+2
                        bytes)
int size)               Maximum number of gun interrupts for 1Vsync (maximum 20)
```

Explanation

Sets up the horizontal/vertical position receive buffer. Retrieval of the horizontal and vertical positions is triggered by an interrupt from the gun.

In order to improve the accuracy of the gun, interrupts and DMAs are blocked within the interrupt handler. Setting a large number of interrupts per 1Vsync consumes 1Hsync of time for each interrupt, so this value should be set low.

Structure of horizontal/vertical position receive buffer:

Table 12-14

Offset	Contents
0	Port number for retrieved horizontal/vertical positions
1	Number of valid horizontal and vertical counters
2,3	Vertical counter value 0
4,5	Horizontal counter value 0
6,7	Vertical counter value 1
8,9	Horizontal counter value 1
...	...
78,79	Vertical counter value 19
80,81	Horizontal counter value 19

(Counter values are half-words (LSB first))

The horizontal counter value returns the system clock value. The pixel value can be obtained by multiplying by the coefficient corresponding to the horizontal resolution shown in the table below.

System clock - pixel clock conversion table:

Table 12-15

Mode	Horizontal resolution	Coefficient
NTSC:	256	0.158532
	320	0.198166
	384	0.226475
	512	0.317065
	640	0.396332
PAL:	256	0.157086
	320	0.196358
	384	0.224409
	512	0.314173
	640	0.392717

[Pixel value] = [Coefficient] x [System clock value] + [Offset]

Horizontal/vertical gun positions are fetched from ports to which a terminal type=3 controller is connected and for guns whose interrupts have been enabled by PadEnableGun().

Gun horizontal/vertical positions can be fetched during each frame, in sequence, beginning with the smallest port number for those ports with guns which are interrupt-enabled.

Check offset 0 in the horizontal/vertical position receive buffer ("Port number for retrieved horizontal/vertical positions") to determine the port number associated with the retrieved horizontal/vertical position.

PadInitGun() is provided only to initialize the gun interrupt environment. In order to communicate with a gun controller, PadInitDirect() or PadInitMtap() must be called first.

In libpad, gun connection state is maintained by the library. If the connection state is invalid, gun position information cannot be obtained. Therefore, as an example, when both parent and child processes use an ID=3 gun, each process must call PadInitGun().

See also

[PadRemoveGun\(\)](#), [PadEnableGun\(\)](#), [PadInitDirect\(\)](#), [PadInitMtap\(\)](#)

PadInitMtap

Initialize controller environment (for Multi Taps).

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

```
void PadInitMtap(
u_char *pad1,          Port 1 receive result (34 bytes)
u_char *pad2)          Port 2 receive result (34 bytes)
```

Explanation

Initializes the control environment for a controller. If a Multi Tap is connected, it is treated as a Multi Tap. If a controller is connected directly to the main PlayStation unit, the structure of the receive buffer is the same as when it is initialized with PadInitDirect().

When using this function, other initialization routines such as PadInitDirect(), InitPAD(), InitGUN(), InitTAP(), and PadInit() cannot be used.

In libpad, controller connection state is maintained by the library. If the connection state is invalid, the controller cannot be recognized. Therefore, when a controller is used by both parent and child processes, each process must call PadInitMtap().

For the format of the receive buffer, see "Receive Buffer Data Format" of Chapter 13 (Controller/Peripherals Library) of the Library Overview.

Note: A Multi Tap may not be recognized if a controller is not connected to port A of the Multi Tap. Therefore, a controller should always be connected to port A of the Multi Tap. This should also be mentioned in the instruction manual.

See also

[PadInitDirect\(\)](#), [PadInitGun\(\)](#), [PadStartCom\(\)](#), [PadStopCom\(\)](#)

PadRead

Read data from the controller (for prototyping only)

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	12/14/98

Syntax

u_long PadRead(
u_short *id*) Controller ID (unused)

Explanation

Reads data from the controller. This function is for prototyping purposes only.

Return value

Controller button status. High 2 bytes are pad 2, low 2 bytes are pad 1.

See also

[PadInit\(\)](#)

PadRemoveGun

Stop retrieval of horizontal/vertical gun position.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

void PadRemoveGun(void)

Explanation

Stops the retrieval of horizontal/vertical gun position information.

See also

[PadInitGun\(\)](#), [PadEnableGun\(\)](#)

PadSetAct

Set transmit buffer.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

void PadSetAct(

int *port*, Target port number (see Explanation)
u_char **data*, Transmit data buffer
int *len*) Length of transmit data buffer (in bytes)

Explanation

Registers the transmit data buffer in the library, so it is not necessary to call this function again if the transmit buffer doesn't change. When the operation of the actuator changes and the contents of the buffer change, the library reads out the buffer every Vsync and automatically transmits the contents to the controller.

port is the target port number, as follows:

Table 12-16

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

When controlling the DUAL SHOCK actuator, not only must the transmit buffer be specified using PadSetAct(), but PadSetActAlign() must also be used to inform the controller of the offset in the transmit buffer where the actuator parameters are located. (The calling sequence of PadSetActAlign() and PadSetAct() is not specified.)

The data length that can be handled by the actuator can be determined with PadInfoAct(). For the SCPH-1200, the data lengths are 1 bit and 1 byte for actuator numbers 0 and 1, respectively. Thus, the actuator can be controlled by using PadSetActAlign() to send the parameter for actuator number 0 at transmit buffer offset 0, and the parameter for actuator number 1 at offset 1. In this case, offset 0 would contain a value of 0 or 1, and offset 1 would contain a value between 0 and 255.

The actuator is stopped when the parameter value is 0, and rotates faster for larger values.

Once the actuator parameters have been sent to the controller during a vertical retrace interrupt, the actuator continues operating even if communication with the controller is suspended by PadEnableCom() or PadStopCom(). However, if communication is suspended for three seconds or more, the controller is reset, at which point the actuator stops operating. Even if the interval during which communication is suspended is less than three seconds, the actuator temporarily stops operating when PadStartCom() reinitiates communication (if communication was suspended with PadStopCom(), it can only be restarted by calling PadStartCom()).

If communication is suspended for three seconds, or if the actuator is halted due to a PadStartCom() read, the value returned from PadState() temporarily becomes PadStateDiscon and a retry is generated to refetch controller information. For this reason, the return value from PadGetState() must be monitored so that refetched actuator information can be properly processed.

If the parameter data length for an actuator is more than one byte, each of the parameter write offsets for that actuator can be set by writing the actuator number in each of the corresponding offsets using PadSetActAlign(). The controller interprets the position of the lowest numbered offset as the high-order byte of the parameter. If the actuator number is written such that the data length for the parameter is exceeded,

the settings beyond the allowed parameter data length are ignored, beginning with the lowest numbered offset.

For a DUAL SHOCK controller, the offsets in the transmit buffer where actuator parameters are written can be specified with `PadSetActAlign()`. However, with analog controller SCPH-1150, there is a pre-determined method for setting up the transmit buffer. The method for setting up the transmit buffer and relevant points to be observed are described below.

Table 12-17

Offset	Contents
0: target device ID	High-order 2 bits: 0x01 Low-order 6 bits: undefined (vibration device)
1: transmit data	bit 0: 1 = vibration ON, 0 = vibration OFF remaining bits (bit 7 ~ 1): undefined
2... : transmit data	Always 0x00. Other values: undefined

Note: The actuator can be turned on only during the 1 Vsync interval before communication with the next controller takes place. During the interval in which the actuator is to be operated, the target device ID should be entered in the transmit data buffer and vibrations should be set to ON at each vertical sync interrupt. The target device ID is valid when the two high-order bits are set to 01. The remaining bits are reserved for the system and should be set to 0. Vibrations are set to ON when the low-order bit of the first transmit data byte is set to 1. The remaining bits should be set to 0 as with the target device ID.

See also

[PadSetActAlign\(\)](#)

PadSetActAlign

Set actuator parameter details to be sent to the controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	2/24/99

Syntax

int PadSetActAlign(

int *port*,

char **data*)

Port number of the controller (see Explanation)

Actuator parameter transmission details (6 bytes)

Explanation

The position in the transmit buffer where the actuator parameters are located is indicated to the controller by writing the actuator numbers in the appropriate positions in the 6-byte array.

port is the port number of the controller to which the actuator parameter details are to be sent:

Table 12-18

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

In the table shown below, offset 0 of the transmit buffer is used for actuator number 0, and offset 1 is used for actuator number 1. The remaining offsets are not used. (The actuator number is entered at positions where transmission is desired, and FF is entered at unused positions.)

Table 12-19

Offset	0	1	2	3	4	5
Contents	00	01	FF	FF	FF	FF

When the Controller ID (return value from `PadInfoMode(port, InfoModeCurExID, 0)`) is different, the number of actuators and the type are also different. When calling `PadSetActAlign()`, the actuator functions from `PadInfoAct()` and the controller ID should be confirmed.

This function doesn't accept requests if the library is communicating with the controller. The return value should be checked to see if the request was accepted. The request is accepted if `PadGetState()` returns `PadStateStable`, so the value from `PadGetState()` can be checked to confirm that the request was accepted. However, when `PadSetActAlign()` and `PadSetMainMode()` are called, the result from `PadGetState()` changes immediately from `PadStateStable` to `PadStateExecCmd`, and `PadSetActAlign()` and `PadSetMainMode()` calls are not accepted until three vertical sync interrupts (six for Multi Taps) have elapsed. Thus, these two functions cannot be called one after the other. If `PadState()` is called instead of checking the return value, `PadGetState()` should be called right before calling the functions to confirm that the return value is `PadStateStable`.

Return value

A 1 is returned if the actuator parameter details request is accepted. 0 is returned if the request is not accepted.

See also

[PadGetState\(\)](#), [PadSetAct\(\)](#), [PadSetMainMode\(\)](#)

PadSetMainMode

Switches / locks the controller mode selector.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

int PadSetMainMode(

int port,

Port number for which the controller mode is to be switched (see Explanation)

int offs,

The controller mode ID table offset which contains the controller mode to be switched

int lock)

If bit 1 is set to 0, the locked/unlocked state of the selector button is kept in its current state. If bit 1 is set to 1 and bit 0 is set to 0, the selector button is unlocked. If bit 1 is 1 and bit 0 is 1, the selector button is locked.

Explanation

Selects the controller mode and switches between locked and unlocked settings for the controller mode selection button on the main controller unit.

port is the port number for which the controller mode is to be switched, as follows:

Table 12-20

	Port 1	Port 2
Direct connection	0x00	0x10
Multi Tap A	0x00	0x10
Multi Tap B	0x01	0x11
Multi Tap C	0x02	0x12
Multi Tap D	0x03	0x13

When this function is called and the controller mode is changed, controller information is retrieved. Therefore, the value returned from `PadGetState()` needs to be monitored so that actuator information can be properly refetched.

This function doesn't accept requests if the library is communicating with the controller. The return value should be checked to see if the request was accepted. The request is accepted if `PadGetState()` returns a `PadStateStable`, so the value from `PadGetState()` can be checked to confirm that the request was accepted. However, when `PadSetMainMode()` or `PadSetActAlign()` is called, the result from `PadGetState()` changes immediately from `PadStateStable` to `PadStateExecCmd`, and `PadSetActAlign()` and `PadSetMainMode()` don't accept requests. Therefore, these two functions cannot be called one after the other. When `PadState()` is checked instead of the return value, `PadGetState()` must be checked right before calling the functions.

Return value

1 when a controller mode setting request was accepted. 0 if the request was not accepted.

See also

[PadGetState\(\)](#), [PadSetActAlign\(\)](#)

PadStartCom

Start reading from controller.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

void PadStartCom(void)

Explanation

Initiates a controller read operation triggered by a vertical retrace interval interrupt.

See also

[PadInitDirect\(\)](#), [PadInitMtap\(\)](#), [PadStopCom\(\)](#), [PadEnableCom\(\)](#)

PadStop

Halt controller (for prototyping only)

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	2.x	12/14/98

Syntax

void PadStop(*void*)

Explanation

Halts all currently connected controllers.

When processing is complete, it is necessary to call this function without fail and halt the controller driver.

This function is for prototyping purposes only.

See also

[PadInit\(\)](#)

PadStopCom

Stop controller read.

Library	Header File	Introduced	Documentation Date
<i>libpad.lib</i>	<i>libpad.h</i>	4.2	12/14/98

Syntax

void PadStopCom(void)

Explanation

Stops a controller read operation. (Stops handling all vertical interval interrupts related to controller services.)

See also

[PadInitDirect\(\)](#), [PadInitMtap\(\)](#), [PadStartCom\(\)](#), [PadEnableCom\(\)](#)

RemoveGUN

Remove gun driver.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.6	12/14/98

Syntax

void RemoveGUN(void)

Explanation

Removes the gun driver registered in InitGUN().

See also

[InitGUN\(\)](#), [StartGUN\(\)](#), [StopGUN\(\)](#), [SelectGUN\(\)](#), [RemoveGUN\(\)](#)

ResetCallback

Initialize all callbacks.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	12/14/98

Syntax

void ResetCallback(void)

Explanation

Initializes all system callbacks. Sets all callback functions to 0 (unregistered), and after securing the interrupt context stack, sets up the environment for accepting interrupts.

ResetCallback() must be called after program boot, before any other processing is performed.

The environment initialized by ResetCallback() remains valid until StopCallback() is called.

It is acceptable to continuously call ResetCallback() without StopCallback(). However, the second and subsequent calls are ignored.

See also

[StopCallback\(\)](#)

RestartCallback

Restart a halted callback.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.2	12/14/98

Syntax

```
int RestartCallback(void)
```

Explanation

Restores the halted call-back to the status immediately prior to when it was halted.

Differs from ResetCallback() in that the call-back functions and call-back stack are not initialized.

ResetCallback() must be executed before executing RestartCallBack().

The environment initialized by RestartCallback() is valid until StopCallback() is called.

There is no problem even if RestartCallback() is successively called without inserting StopCallback(), but calls from the second one onwards are ignored.

Return value

Used by system only.

See also

[StopCallback\(\)](#)

SelectGUN

Select gun.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.5	12/14/98

Syntax

void SelectGUN(

int *ch*, Gun channel (0 or 1)

u_char *mask*) Interruptmask setting (0: interrupts prohibited, 1: interrupts permitted)

Explanation

Sets the interrupt mask for the gun.

It is not possible to disable interrupts for two masks at the same time.

See also

[InitGUN\(\)](#), [StartGUN\(\)](#), [StopGUN\(\)](#), [RemoveGUN\(\)](#)

SetVideoMode

Declare current video signaling system.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.1	12/14/98

Syntax

```
long SetVideoMode(  
long mode)           Video signaling system mode
```

Explanation

Declares the video signaling system indicated by *mode* to the libraries (MODE_NTSC for NTSC and MODE_PAL for PAL).

Related libraries will conform to the actions of the declared video signaling system environment.

Should be called in advance of all library functions.

Return value

Previously-set video signaling system mode.

See also

[GetVideoMode\(\)](#)

StartGUN

Start controller reading.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.6	12/14/98

Syntax

long StartGUN(*void*)

Explanation

Starts controller reading at Vsync interrupt.

Return value

1 if successful; 0 on failure.

See also

[InitGUN\(\)](#), [StopGUN\(\)](#), [SelectGUN\(\)](#), [RemoveGUN\(\)](#)

StartTAP

Start controller reading.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.4	12/14/98

Syntax

void StartTAP(*void*)

Explanation

Starts controller reading at Vsync interrupt.

See also

[InitTAP\(\)](#)

StopCallback

Stop all callbacks.

Library	Header File	Introduced	Documentation Date
<i>libetc.lib</i>	<i>libetc.h</i>	3.0	12/14/98

Syntax

void StopCallback(void)

Explanation

Stops all system callbacks.

Before terminating programs, StopCallback() can be called to disable all interrupts.

See also

[RestartCallback\(\)](#)

StopGUN

Halt controller reading.

Library	Header File	Introduced	Documentation Date
<i>libgun.lib</i>	<i>libgun.h</i>	3.6	12/14/98

Syntax

void StopGUN(void)

Explanation

Halts the controller reading. Does not prohibit interrupts.

See also

[InitTAP\(\)](#), [StartGUN\(\)](#), [SelectGUN\(\)](#), [RemoveGUN\(\)](#)

StopTAP

Halt controller reading.

Library	Header File	Introduced	Documentation Date
<i>libtap.lib</i>	<i>libtap.h</i>	3.4	12/14/98

Syntax

void StopTAP(*void*)

Explanation

Halts the controller reading. Does not prohibit interrupts.

See also

[InitTAP\(\)](#)

Chapter 13: Link Cable Library

Table of Contents

Functions

_comb_control	13-3
AddCOMB	13-5
ChangeClearSIO	13-6
DelCOMB	13-7

Macros

CombAsyncRequest	13-8
CombBytesRemaining	13-9
CombBytesToRead	13-10
CombBytesToWrite	13-11
CombCancelRead	13-12
CombCancelWrite	13-13
CombControlStatus	13-14
CombCTS	13-15
CombGetBPS	13-16
CombGetMode	13-17
CombGetPacketSize	13-18
CombReset	13-19
CombResetError	13-20
CombResetVBLANK	13-21
CombSetBPS	13-22
CombSetControl	13-23
CombSetMode	13-24
CombSetPacketSize	13-25
CombSetRTS	13-26
CombSioStatus	13-27
CombWaitCallback	13-28

_comb_control

Link cable driver control.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	12/14/98

Syntax

```
long _comb_control(
u_long cmd           Command
u_long arg           Subcommand
u_long param)        Argument
```

Explanation

Offers the same functionality as `ioctl()` to an SIO device.

All the macros in this chapter are versions of this command.

Table 13-1: _comb_control() Command Summary

cmd	arg	Function
0	0	Returns the serial controller status (see Table 13-2)
0	1	Returns the control line status (see Table 13-3)
0	2	Returns the communication mode (see Table 13-4)
0	3	Returns the communication rate in bps
0	4	Returns the "unit-number of characters for receiving"
0	5	Returns the amount of remaining data (bytes) from asynchronous input/output during processing
		If the param is 0 it is asynchronous write, if 1 it is asynchronous read
0	6	Returns an asynchronous input/output request whether it registered or not
		If it has been registered, it will return 1. Others will return 0.
		If the param is 0, it is asynchronous write, if 1 it is asynchronous read
1	0	System reserved
1	1	Sets the value of param as the control line status (*2)
1	2	(Reserved)
1	3	Sets the value of param as the communication rate by bps
1	4	Sets the value of param as the "unit-number of characters for receiving"
2	0	Resets the serial controller
		Controller status, communication mode and communication speed are saved
2	1	Clears the bits related to the driver status error.
		Includes a function which indicates the completion of the interrupt processing to the driver
2	2	Cancels the asynchronous writing
2	3	Cancels the asynchronous reading
3	0	When param is 1 RTS is made 1
		When param is 0, RTS is made 0
3	1	If (CTS==1) 1 is returned, the others return 0
4	0	The param value is considered to be the pointer to the function and is registered as the pointer to the wait callback function
		The callback function pointer values up to that point are returned

Table 13-2: Driver Status

bit	Contents
31-10	Undefined
9	1: Interrupt is ON
8	1: CTS is ON
7	1: DSR is ON
6	Undefined
5	1: Frame error occurrence
4	1: Overrun error occurrence
3	1: Parity error occurrence
2	1: No sending data
1	1: Possible to read the receiving data
0	1: Possible to write the sending data

Table 13-3: Control Line Status

bit	Contents
31-2	Undefined
1	1: RTS is ON
0	1: DTR is ON

Table 13-4: Communication Mode

bit	Contents
31-8	Undefined
7,6	Stop bit length 01:1 10:1.5 11:2
5	Parity check(2) 1: odd number 0: even number
4	Parity check(1) 1: enabled
3,2	Character length 00:5 bits 01:6 10:7 11:8
1	1 at all times
0	0 at all times

Return value

Depends on the control command *cmd*.

AddCOMB

Initialize link cable driver.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	12/14/98

Syntax

void AddCOMB(*void*)

Explanation

Initializes the link cable driver.

See also

[DelCOMB\(\)](#)

ChangeClearSIO

Clear interrupt from expanded SIO in the driver.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	12/14/98

Syntax

```
void ChangeClearSIO(  
long val)           Interrupt cause clear flag
```

Explanation

If *val* is non-0, an interrupt from an expansion SIO in the driver is cleared. This is used only when other expansion SIO drivers are also present.

DelCOMB

Remove link cable driver from kernel.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	3.0	12/14/98

Syntax

void DelCOMB(*void*)

Explanation

Removes link cable driver from kernel.

See also

[AddCOMB\(\)](#)

CombAsyncRequest

Get asynchronous communication request status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombAsyncRequest(

long *param*) 0: asynchronous write, 1: asynchronous read

Explanation

Determines whether an asynchronous input/output request has been made.

This macro is equivalent to `_comb_control (0, 6, param)`.

Return value

1 if request has been made; 0 otherwise.

CombBytesRemaining

Get remaining transmit or receive data.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

**long CombBytesRemaining(
long *param*)** 0: asynchronous write, 1: asynchronous read

Explanation

Gets the remaining data count from the asynchronous read or asynchronous write being processed.

This macro is equivalent to `_comb_control (0, 5, param)`.

Return value

The number of bytes remaining.

See also

[_comb_control\(\)](#)

CombBytesToRead

Get number of bytes left to receive.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombBytesToRead(void)

Explanation

Obtains the number of bytes left in the current asynchronous read operation.

This macro is equivalent to `_comb_control (0, 5, 1)`.

Return value

The number of bytes remaining.

See also

[_comb_control\(\)](#)

CombBytesToWrite

Get number of bytes left to send.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombBytesToWrite(void)

Explanation

Obtains the number of bytes remaining in the current asynchronous write operation.

This macro is equivalent to `_comb_control (0, 5, 0)`.

Return value

The number of bytes remaining.

See also

[_comb_control\(\)](#)

CombCancelRead

Cancel asynchronous read.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombCancelRead(void)

Explanation

Cancels current asynchronous read operation.

This macro is equivalent to `_comb_control (2, 3, 0)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombCancelWrite

Cancel asynchronous write.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombCancelWrite(void)

Explanation

Cancels current asynchronous write operation.

This macro is equivalent to `_comb_control (2, 2, 0)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombControlStatus

Get control line status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombControlStatus(void)

Explanation

Obtains the control line status.

This macro is equivalent to `_comb_control (0, 1, 0)`.

Return value

The control line status. Bit fields are as follows:

Table 13-5: Control Line Status

bit	Contents
31-2	undefined
1	1: RTS on
0	1: DTR on

See also

[_comb_control\(\)](#)

CombCTS

Get status of CTS signal.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombCTS(void)

Explanation

Obtains the state of the serial controller CTS bit.

This macro is equivalent to `_comb_control (3, 1, 0)`.

Return value

1 if CTS is 1; 0 otherwise.

See also

[_comb_control\(\)](#)

CombGetBPS

Get communication speed.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombGetBPS(void)

Explanation

Obtains the communication speed (in bps).

This macro is equivalent to `_comb_control (0, 3, 0)`.

Return value

The communication speed (in bps).

See also

[_comb_control\(\)](#)

CombGetMode

Get communication mode.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombGetMode(void)

Explanation

Obtains the communication mode.

This macro is equivalent to `_comb_control (0, 2, 0)`.

Return value

The communication mode.

Table 13-6: Communication Mode

bit	Contents
31-8	undefined
7,6	stop bit length 01: 1 10: 1.5 11: 2
5	parity2 1:odd 0:even
4	parity1 1:enabled
3,2	character length 00: 5 bits 01: 6 10: 7 11: 8
1	always 1
0	always 0

See also

[_comb_control\(\)](#)

CombGetPacketSize

Get receive packet size.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombGetPacketSize(void)

Explanation

Obtains the receive packet size.

This macro is equivalent to `_comb_control (0, 4, 0)`.

Return value

The receive packet size.

See also

[_comb_control\(\)](#)

CombReset

Initialize the serial controller.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombReset(void)

Explanation

Initializes the serial controller. Controller status, communication mode and communication speed remain unchanged.

This macro is equivalent to `_comb_control (2, 0, 0)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombResetError

Initialize error flags.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombResetError(void)

Explanation

Clears error-related bits from driver status.

This macro is equivalent to `_comb_control (2, 1, 0)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombResetVBLANK

Reset vertical blanking signal.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombResetVBLANK(*void*)

Explanation

Resets the vertical blanking signal.

This macro is equivalent to `_comb_control (5, 0, 0)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombSetBPS

Set communication speed.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombSetBPS(
long *bps* Communication speed (in bps)

Explanation

Sets the communication speed. *bps* must be in the range 9600 - 2073600 and evenly divisible into 2073600. If asynchronous write is used, the maximum communication speed is 57600 bps.

This macro is equivalent to `_comb_control(1, 3, bps)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombSetControl

Set control line status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

```
long CombSetControl(
long val)           Control line status
```

Explanation

Sets the control line status.

Table 13-7: Control Line Status

bit	Contents
31-2	unused
1	1: RTS on
0	1: DTR on

This macro is equivalent to `_comb_control (1, 1, val)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombSetMode

Set communication mode.

Library	Header File	Introduced	Documentation Date
libcomb.lib	libcomb.h	4.2	12/14/98

Syntax

long CombSetMode(
long mode) Communication mode

Explanation

Sets the communication mode.

This macro is equivalent to _comb_control (1, 2, mode).

Table 13-8: Communication Mode

bit	Contents
31-8	Unused
7,6	stop bit length 01: 1 10: 1.5 11: 2
5	Parity2 1: odd 0: even
4	Parity1 1: enabled
3,2	Character length 00: 5 bits 01: 6 10: 7 11: 8
1	Always 1
0	Always 0

Return value

0.

See also

[_comb_control\(\)](#)

CombSetPacketSize

Set receive packet size.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

**long CombSetPacketSize(
long *size*)** Packet size (1, 2, 4, or 8)

Explanation

Sets the receive packet size, which sets the byte count used for generating interrupts in asynchronous communication. For example, if the receive packet size is set to 4, the serial controller generates an interrupt after every four bytes of data received. A large packet size lowers the frequency of interrupts, thus improving overall system performance.

Note: When sending data asynchronously, the packet size must be set to 1, since only 1 byte can be sent at a time.

This macro is equivalent to `_comb_control (1, 4, size)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombSetRTS

Set RTS signal.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombSetRTS(void)

Explanation

Sets the RTS bit in control line status to 1.

This macro is equivalent to `_comb_control (3, 0, 1)`.

Return value

0.

See also

[_comb_control\(\)](#)

CombSioStatus

Get serial controller status.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

long CombSioStatus(void)

Explanation

Obtains serial controller status.

This macro is equivalent to `_comb_control (0, 0, 0)`.

Return value

The serial controller status. Bit fields are:

Table 13-9: Serial Controller Status

Bit	Contents
31-10	undefined
9	1: interrupts on
8	1: CTS is on
7	1: DSR is on
6	undefined
5	1:frame error generated
4	1:overflow error generated
3	1:parity error generated
2	1:no data to transmit
1	1:receive data available
0	1:transmit data available

See also

[_comb_control\(\)](#)

CombWaitCallback

Set wait callback function.

Library	Header File	Introduced	Documentation Date
<i>libcomb.lib</i>	<i>libcomb.h</i>	4.2	12/14/98

Syntax

**long CombWaitCallback(
long *func*)** Pointer to the wait callback function

Explanation

The value of *func* is entered as a pointer to the wait callback function.

This macro is equivalent to `_comb_control (4, 0, func)`.

Return value

The value of the previous callback function.

See also

[_comb_control\(\)](#)

Chapter 14: Extended Sound Library

Table of Contents

Structures

ProgAtr	14-5
SndRegisterAttr	14-6
SndVoiceStats	14-7
SndVolume	14-8
SndVolume2	14-9
VabHdr	14-10
VagAtr	14-11
_SsFCALL	14-12

Functions

dmy_Ss...	14-14
SsAllocateVoices	14-15
SsBlockVoiceAllocation	14-16
SsChannelMute	14-17
SsEnd	14-18
SsGetActualProgFromProg	14-19
SsGetChannelMute	14-20
SsGetCurrentPoint	14-21
SsGetMute	14-22
SsGetMVol	14-23
SsGetNck, SsSetNck, SsSetNoiseOff, SsSetNoiseOn	14-24
SsGetRVol	14-25
SsGetSerialAttr	14-26
SsGetSerialVol	14-27
SsGetVoiceMask	14-28
SsInit	14-29
SsInitHot	14-30
SsIsEos	14-31
SsPitchFromNote	14-32
SsPlayBack	14-33
SsQueueKeyOn	14-34
SsQueueRegisters	14-35
SsQueueReverb	14-36
SsQuit	14-37
SsSepClose	14-38
SsSepOpen	14-39
SsSepOpenJ	14-40
SsSepPause	14-41
SsSepPlay	14-42
SsSepReplay	14-43
SsSepSetAccelerando	14-44
SsSepSetCrescendo	14-45
SsSepSetDecrescendo	14-46
SsSepSetRitardando	14-47
SsSepSetVol	14-48
SsSepStop	14-49
SsSeqCalledTbyT	14-50
SsSeqClose	14-51
SsSeqGetVol	14-52
SsSeqOpen	14-53
SsSeqOpenJ	14-54
SsSeqPause	14-55
SsSeqPlay	14-56

SsSeqPlayPtoP	14-57
SsSeqReplay	14-58
SsSeqSetAccelerando	14-59
SsSeqSetCrescendo	14-60
SsSeqSetDecrescendo	14-61
SsSeqSetNext	14-62
SsSeqSetRitardando	14-63
SsSeqSetVol	14-64
SsSeqSkip	14-65
SsSeqStop	14-66
SsSetAutoKeyOffMode	14-67
SsSetCurrentPoint	14-68
SsSetLoop	14-69
SsSetMarkCallback	14-70
SsSetMono	14-71
SsSetMute	14-72
SsSetMVol	14-73
SsSetNext	14-74
SsSetReservedVoice	14-75
SsSetRVol	14-76
SsSetSerialAttr	14-77
SsSetSerialVol	14-78
SsSetStereo	14-79
SsSetTableSize	14-80
SsSetTempo	14-81
SsSetTickCallback	14-82
SsSetTickMode	14-83
SsSetVoiceMask	14-84
SsSetVoiceSettings	14-85
SsStart	14-86
SsStart2	14-87
SsUnBlockVoiceAllocation	14-88
SsUtAllKeyOff	14-89
SsUtAutoPan	14-90
SsUtAutoVol	14-91
SsUtChangeADSR	14-92
SsUtChangePitch	14-93
SsUtFlush	14-94
SsUtGetDetVVol	14-95
SsUtGetProgAtr	14-96
SsUtGetReverbType	14-97
SsUtGetVabHdr	14-98
SsUtGetVagAddr	14-99
SsUtGetVagAddrFromTone	14-100
SsUtGetVagAtr	14-101
SsUtGetVBaddrInSB	14-102
SsUtGetVVol	14-103
SsUtKeyOff	14-104
SsUtKeyOffV	14-105
SsUtKeyOn	14-106
SsUtKeyOnV	14-107
SsUtPitchBend	14-108
SsUtReverbOff	14-109
SsUtReverbOn	14-110
SsUtSetDetVVol	14-111
SsUtSetProgAtr	14-112
SsUtSetReverbDelay	14-113

SsUtSetReverbDepth	14-114
SsUtSetReverbFeedback	14-115
SsUtSetReverbType	14-116
SsUtSetVabHdr	14-117
SsUtSetVagAtr	14-118
SsUtSetVVol	14-119
SsVabClose	14-120
SsVabFakeBody	14-121
SsVabFakeHead	14-122
SsVabOpen	14-123
SsVabOpenHead	14-124
SsVabOpenHeadSticky	14-125
SsVabTransBody	14-126
SsVabTransBodyPartly	14-127
SsVabTransCompleted	14-128
SsVabTransfer	14-129
SsVoiceCheck	14-130
SsVoKeyOff	14-131
SsVoKeyOn	14-132

ProgAtr

Program header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Structure

typedef struct ProgAtr {

u_char <i>tones</i> ;	Number of VAG attribute sets contained in the program
u_char <i>mvol</i> ;	Master volume for the program
u_char <i>prior</i> ;	Program priority (0-15)
u_char <i>mode</i> ;	Sound source mode
u_char <i>mpan</i> ;	Program pan
char <i>reserved0</i> ;	Reserved by the system
short <i>attr</i> ;	Program attribute
u_long <i>reserved1</i> ;	Reserved by the system
u_long <i>reserved2</i> ;	Reserved by the system

};

SndRegisterAttr

SPU register attributes.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

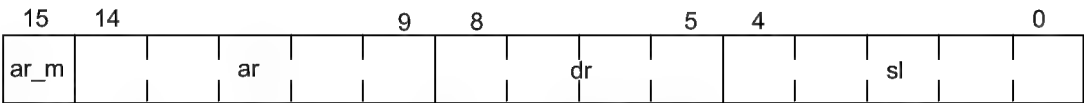
Structure

```
struct SndRegisterAttr {
    SndVolume2 volume;      Volume data for left and right channels
    short pitch;            Pitch rate at which to play back waveform data
    short mask;             Bitfield designating which attributes to set
    short addr;             Waveform data start address
    short adsr1;            Bitfield for setting adsr information (see Explanation)
    short adsr2;            Bitfield for setting adsr information (see Explanation)
} SndRegisterAttr;
```

Explanation

This structure is used in the function SsQueueRegisters() to set SPU voice information.

adsr1:



ar_m: attack rate mode 0 = linear, 1 = exponential
ar: attack rate
dr: decay rate
sl: sustain level

adsr2 :



sr_m: sustain rate mode 0 = linear, 1 = exponential
sr_s: sustain rate sign 0 = positive, 1 = negative
sr: sustain rate
rr_m: release rate mode 0 = linear, 1 = exponential
rr: release rate

Note: Bit 13 is unused

See also

[SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SndVoiceStats

Internal libsnd voice variables.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Structure

```
struct SndVoiceStats {
    short vagld;           VAG number pointed to by tone information (1-254)
    short vabld;           VAB number containing tone information (0-15)
    u_short pitch;         Playback rate of voice
    short vol;             Volume of voice (0-127). Not valid for 3D sound input
    char pan;              Voice pan (0-127; 0 = left, 64 = center, 127 = right) . Not valid for 3D sound
                           input
    short note;            Note at which tone information keyed on
    short tone;            Tone number (0-15)
    short prog_num;        Program number containing tone information (0-127)
    short prog_actual;     The "real" program number within which the tone information resides.
} SndVoiceStats;
```

Explanation

This structure is used to fill the internal libsnd voice structures in the function `SsSetVoiceSettings()`.

prog_actual is only incremented by valid programs (programs containing one or more tones) and so may differ from *prog_num*. Example: In a VAB with valid programs 0-10 and 100-127, the *prog_num* of program 127 = 127, while the *prog_actual* of program 127 = 38. Used to calculate offset in VAB header of tone information.

See also

[SndRegisterAttr\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SndVolume

Volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Structure

typedef struct

SndVolume {

u_short *left*;

u_short *right*;

};

L channel volume value, 0 - 127

R channel volume value, 0 - 127

SndVolume2

Volume-greater range.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Structure

```
struct SndVolume2 {
    short left;           Left volume value, -0x4000 ~ 0x3fff
    short right;          Right volume value, -0x4000 ~ 0x3fff
} SndVolume2;
```

Explanation

This structure allows for a greater range of volume inputs, including negative volumes, when used with the libsnd keyon emulation series: SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters() -> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation().

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

VabHdr

Bank header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Structure

```
typedef struct VabHdr {
    long form;           Format name (always 'VABp')
    long ver;            Format version number
    long id;             Bank (VAB) number
    u_long fsize;        Bank file size
    u_short reserved0;   Reserved by the system
    u_short ps;          Total number of programs contained in the bank
    u_short ts;          Total number of tones contained in the bank
    u_short vs;          Number of VAGs contained in the bank
    u_char mvol;         Master volume
    u_char pan;          Master pan level
    u_char attr1;        Bank attribute 1 that can be defined by the user
    u_char attr2;        Bank attribute 2 that can be defined by the user
    u_long reserved1;    Reserved by the system
};
```

Explanation

The VAB bank header contains information, such as sound source data set size and sound source numerals, that is used at the time of execution.

When `SsVabOpenHead()` is called, it is read by the system and wave form data is generated in the SPU's local memory. Also, volume setting and panning setting are referred at the time of voice allocation.

Information about VAB, the program and each VAG header can change at the time of execution by the user, and the attribute value is reflected in the voice application after the next KEY ON.

VagAtr

Waveform header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Structure

typedef struct VagAtr {

u_char <i>prior</i> ;	Priority (0-15)
u_char <i>mode</i> ;	Sound source mode (Bit values 0: normal, 4: reverb)
u_char <i>vol</i> ;	Volume (0-127, 0:min, 127:max)
u_char <i>pan</i> ;	Pan pot (0-127, 0:left, 63:center, 127:right)
u_char <i>center</i> ;	Center note (0-127)
u_char <i>shift</i> ;	Pitch correction (0-127, in cents) (center note fine tune)
u_char <i>min</i> ;	Minimum note limit
u_char <i>max</i> ;	Maximum note limit
u_char <i>vibW</i> ;	Vibrato width (0-127 over one octave) (not implemented)
u_char <i>vibT</i> ;	Period of vibrato cycle (in ticks) (not implemented)
u_char <i>porW</i> ;	Portamento width (not implemented)
u_char <i>porT</i> ;	Period of portamento duration (in ticks) (not implemented)
u_char <i>pbmin</i> ;	Minimum pitch bend limit
u_char <i>pbmax</i> ;	Maximum pitch bend limit
u_char <i>reserved1</i> ;	Reserved by the system
u_char <i>reserved2</i> ;	Reserved by the system
u_short <i>adsr1</i> ;	Set ADSR value 1
u_short <i>adsr2</i> ;	Set ADSR value 2
short <i>prog</i> ;	Master program containing the VAG attribute
short <i>vag</i> ;	VAG's ID number utilized by the VAG attribute
short <i>reserved[4]</i> ;	Reserved by the system

};

_SsFCALL

Function table type referenced in SsSeqOpenJ() and SsSepOpenJ().

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	12/14/98

Structure

```
typedef struct {
    void (*noteon) ();
    void (*programchange) ();
    void (*pitchbend) ();
    void (*metaevent) ();
    void (*control[13]) ();
    void (*ccentry[20]) ();
} _SsFCALL;
```

Members

All members hold pointers to low-level MIDI functions.

Table 14-1

Member	Pointer to	
	<i>_SsNoteOn</i>	
	<i>_SsSetProgramChange</i>	
	<i>_SsSetPitchBend</i>	
	<i>_SsGetMetaEvent</i>	
<i>Specified when using</i>	<i>_SsSetControlChange</i>	
<i>Control Change</i>		
<i>Control Change #1 (Bank</i>	<i>_SsContBankChange</i>	
<i>Change)</i>		
<i>Control Change #6 (Data</i>	<i>_SsContDataEntry</i>	
<i>Entry)</i>		
<i>Control Change #7 (Main</i>	<i>_SsContMainVol</i>	
<i>Volume)</i>		
<i>Control Change #10 (Pan</i>	<i>_SsContPanpot</i>	
<i>Pot)</i>		
<i>Control Change #11</i>	<i>_SsContExpression</i>	
<i>(Expression)</i>		
<i>control [CC_DAMPER]</i>	<i>_SsContDamper</i>	Control Change #64 (Damper pedal)
<i>control [CC_NRPN1]</i>	<i>_SsContNrpn1</i>	Control Change #98 (NRPN)
<i>control [CC_NRPN2]</i>	<i>_SsContNrpn2</i>	Control Change #99 (NRPN)
<i>control [CC_RPN1]</i>	<i>_SsContRpn1</i>	Control Change #100 (RPN)
<i>control [CC_RPN2]</i>	<i>_SsContRpn2</i>	Control Change #101 (RPN)
<i>control [CC_EXTERNAL]</i>	<i>_SsContExternal</i>	Control Change #91 (External Effect Depth)
<i>control [CC_RESETALL]</i>	<i>_SsContResetAll</i>	Control Change #121 (Reset All)
<i>ccentry [DE_PRIORITY]</i>	<i>_SsSetNrpnVabAttr0</i>	priority
<i>ccentry [DE_MODE]</i>	<i>_SsSetNrpnVabAttr1</i>	mode
<i>ccentry [DE_LIMITL]</i>	<i>_SsSetNrpnVabAttr2</i>	limit low
<i>ccentry [DE_LIMITH]</i>	<i>_SsSetNrpnVabAttr3</i>	limit high
<i>ccentry [DE_ADSR_AR_L]</i>	<i>_SsSetNrpnVabAttr4</i>	ADSR (AR-L)
<i>ccentry [DE_ADSR_AR_E]</i>	<i>_SsSetNrpnVabAttr5</i>	ADSR (AR-E)
<i>ccentry [DE_ADSR_DR]</i>	<i>_SsSetNrpnVabAttr6</i>	ADSR (DR)

Member	Pointer to	
<i>ccentry</i> [DE_ADSR_SL]	<i>_SsSetNrpnVabAttr7</i>	ADSR (SL)
<i>ccentry</i> [DE_ADSR_SR_L]	<i>_SsSetNrpnVabAttr8</i>	ADSR (SR-L)
<i>ccentry</i> [DE_ADSR_SR_E]	<i>_SsSetNrpnVabAttr9</i>	ADSR (SR-E)
<i>ccentry</i> [DE_ADSR_RR_L]	<i>_SsSetNrpnVabAttr10</i>	ADSR (RR-L)
<i>ccentry</i> [DE_ADSR_RR_E]	<i>_SsSetNrpnVabAttr11</i>	ADSR (RR-E)
<i>ccentry</i> [DE_ADSR_SR]	<i>_SsSetNrpnVabAttr12</i>	ADSR (SR)
<i>ccentry</i> [DE_VIB_TIME]	<i>_SsSetNrpnVabAttr13</i>	vibrate time (no support)
<i>ccentry</i> [DE_PORTA_DEPTH]	<i>_SsSetNrpnVabAttr14</i>	portamento depth (no support)
<i>ccentry</i> [DE_REV_TYPE]	<i>_SsSetNrpnVabAttr15</i>	reverb type
<i>ccentry</i> [DE_REV_DEPTH]	<i>_SsSetNrpnVabAttr16</i>	reverb depth
<i>ccentry</i> [DE_ECHO_FB]	<i>_SsSetNrpnVabAttr17</i>	echo feedback
<i>ccentry</i> [DE_ECHO_DELAY]	<i>_SsSetNrpnVabAttr18</i>	echo delay
<i>ccentry</i> [DE_DELAY]	<i>_SsSetNrpnVabAttr19</i>	delay time

Explanation

The functions `SsSeqPlay()` and `SsSepPlay()` analyze the MIDI status data and call low-level functions. When calling `SsSeqOpen()` or `SsSepOpen()`, all the low-level functions are linked in, even though an application won't necessarily use them all.

The new functions `SsSeqOpenJ()` and `SsSepOpenJ()` have been added to replace `SsSeqOpen()` and `SsSepOpen()` respectively. With these functions, all the low-level functions are in a jump table, so the user can select only the desired function groups. Unnecessary functions aren't linked, so code size is reduced.

The `_SsFCALL` structure defines this function table. Low-level functions that have pointers assigned to them are linked in. Low-level functions can be eliminated by not setting the member.

To determine which functions will be called by a MIDI sequence, you can use a test program. This is necessary because, if a MIDI sequence calls a low-level function which hasn't been linked in, a BUS ERROR results. Set all pointers for low-level functions to the correspondingly named `dmy_Ss...()` function. Each low-level function called by the MIDI sequence outputs a message via `printf()`.

`SsFCALL` is the name of the actual `libsnd` variable that must be used by the programmers to link the low-level MIDI functions.

See also

[dmy_Ss...\(\)](#), [SsSeqOpenJ\(\)](#), [SsSepOpenJ\(\)](#)

dmy_Ss...

Test function for low-level MIDI jump table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	12/14/98

Syntax

void dmy_Ss...(void)

Explanation

Hook these dummy functions into the libsnd variable SsFCALL (structure type _SsFCALL).

Ex: SsFCALL.noteon = (void (*)()) dmy_Ss_NoteOn();

When these functions are called for the first time, the name of the low-level MIDI function is output by printf(). After all of the low-level MIDI functions that need to be called by your program have been determined, replace the registered dmy_Ss...() calls with the appropriate _Ss...() calls. Unused dmy_Ss...() should be deleted.

This function is provided for debugging.

SsAllocateVoices

Compare priorities of a number of voices and allocate them where possible.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

long SsAllocateVoices(

u_char *voices*,

u_char *priority*)

The desired number of voices required to be keyed on simultaneously

Priority of the desired voices

Range: 0-127, with 0 being lowest priority and 127 being highest priority

Explanation

Emulates the libsnd voice allocation system, but allows more than one voice to be allocated simultaneously. Searches through all 24 SPU voices for SPU voices in the state SPU_OFF, ENV_OFF (that is, SPU voices which are not currently sounding). If there are fewer SPU voices in a state of SPU_OFF, ENV_OFF than the total number of desired *voices*, the levels of voice priority are compared, and the lowest priority SPU voice number is allocated for the desired *voices* (if the lowest priority is less than the value set in *priority*).

Where the priorities are equivalent, the SPU voice number with the lowest envelope is allocated to the desired *voices*.

Where the priorities and envelope size are the same, the oldest SPU voices are allocated to the desired *voices*.

This function should only be used as part of the series: SsBlockVoiceAllocation() -> SsAllocateVoices()-> SsSetVoiceSettings()-> SsQueueRegisters()-> SsQueueKeyOn()-> SsQueueReverb() -> SsUnBlockVoiceAllocation()

Return value

A bifield specifying which voices were allocated for key on. To determine if a voice was allocated, AND the return value with the mask for a particular voice (SPU_00CH ...SPU_23CH). If the value is non-zero, the voice was allocated.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SsBlockVoiceAllocation

Block voice allocation system used by SsUtKeyOn(), SsUtKeyOnV(), and MIDI key on commands.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

char SsBlockVoiceAllocation(void)

Explanation

Blocks the voice allocation system for libsnd functions SsUtKeyOn() and SsUtKeyOnV(). Once this function is called, those functions will return -1. MIDI key on commands are also blocked until SsUnBlockVoiceAllocation() is called, in order to ensure proper key on.

The time spent until SsUnBlockVoiceAllocation() should be short, in order to reduce missed key on commands.

This function should only be used as part of the series: SsBlockVoiceAllocation() -> SsAllocateVoices()-> SsSetVoiceSettings()-> SsQueueRegisters()-> SsQueueKeyOn()-> SsQueueReverb() -> SsUnBlockVoiceAllocation()

Return value

1 if successful; -1 if voice allocation system already blocked, by either a previous call to this function with no corresponding call to SsUnBlockVoiceAllocation() or a call to SsUtKeyOn(), SsUtKeyOnV() or a MIDI key on command.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsSetVoiceSettings\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueReverb\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsFindPitch\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsVoiceCheck\(\)](#)

SsChannelMute

Select MIDI channels for muting.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	12/14/98

Syntax

void SsChannelMute(

short *acn*, SEP access number
short *trn*, SEQ number within SEP
 (0 when the music score data is SEQ)
long *channels*) MIDI channel

Explanation

Selects MIDI channels that are muted. The low 16 bits of *channels* represent each channel; a bit set to 1 means the voice should be muted. This function can be called when playing is in progress, or before playing has begun, to initiate muting.

See also

[SsSeqPlay\(\)](#), [SsSepPlay\(\)](#)

SsEnd

Stop the sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsEnd(*void*)

Explanation

If `SsSetTickMode()` is used to set a mode that automatically calls `SsSeqCalledTbyT()`, this function stops `SsSeqCalledTbyT()` from being called at every tick.

See also

[SsStart\(\)](#), [SsSetTickMode\(\)](#), [SsSeqCalledTbyT\(\)](#), [SsQuit\(\)](#)

SsGetActualProgFromProg

Convert a program number into a “real” program or offset number.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

```
short SsGetActualProgFromProg(
short vabId,           VAB number containing desired tone information
short ProgNum)       Program number containing tone information
```

Explanation

Used to determine the “real” program number of tone information. This number is only incremented by valid programs (programs containing one or more tones) and so may differ from the program number. Example: In a VAB with valid programs 0-10 and 100-127, the *prog_num* of program 127 = 127, while the *prog_actual* of program 127 = 38. This number is used to calculate the offset of tone information in the VAB header.

Return value

The “real” program number upon success; -1 if *vabId* or *ProgNum* are out of range.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SsGetChannelMute

Get muted channel number

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.0	12/14/98

Syntax

long SsGetChannelMute(

short *sep_num*,

SEQ/SEP access number

short *seq_num*)

SEQ number within SEP data

Explanation

Returns muted MIDI channels.

Return value

Bit field showing muted MIDI channels (1= muted; 0 = not muted).

See also

[SsChannelMute\(\)](#)

SsGetCurrentPoint

Get current position in SEQ/SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	12/14/98

Syntax

```
u_char *SsGetCurrentPoint(
short acn,           SEP access number
short tm)           SEQ number within SEP
                    (0 when the music score data is SEQ)
```

Explanation

Obtains the address of the current position in the SEQ/SEP data that is being played.

Return value

SEP/SEQ data address.

See also

[SsSeqPlay\(\)](#), [SsSepPlay\(\)](#)

SsGetMute

Get mute attribute.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

char SsGetMute (*void*)

Explanation

Obtains the mute attribute.

Return value

SS_MUTE_ON = Mute on; SS_MUTE_OFF = Mute off

See also

[SsSetMute\(\)](#)

SsGetMVol

Get main volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
void SsGetMVol(  
  SndVolume *m_vol)      Pointer to main volume value
```

Explanation

Returns the main volume value to *m_vol*.

See also

[SsSetMVol\(\)](#)

SsGetNck, SsSetNck, SsSetNoiseOff, SsSetNoiseOn

Libsnd noise functions (Not supported)

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	12/14/98

Syntax

short SsGetNck(void) Get noise clock value

void SsSetNck(
short *n_clock*) Set noise clock value
 Noise clock value (0 – 0x3f)

void SsSetNoiseOff(void) Sets noise off

void SsSetNoiseOn(
short *voll*, Sets noise on
 L channel volume value
short *volr*) R channel volume value

Explanation

Libsnd noise functions.

Note: These functions are not supported. Instead, use libspu noise functions or create a noise VAG from recorded AIFF.

SsGetRVol

Get reverb volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

**void SsGetRVol(
[SndVolume](#) *r_vol)** Pointer to reverb volume value

Explanation

Returns the reverb volume value to *r_vol*.

See also

[SsSetRVol\(\)](#)

SsGetSerialAttr

Get value of a serial attribute.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

char SsGetSerialAttr(

char *s_num*, Serial Number

char *attr*) Attribute

Explanation

Returns the specified serial attribute value.

s_num can be SS_SERIAL_A for Serial A (CD input), or SS_SERIAL_B for Serial B (external digital input).

attr can be SS_MIX for mixing, or SS_REV for reverb.

Return value

1 if attribute is on, 0 if attribute is off.

See also

[SsSetSerialAttr\(\)](#)

SsGetSerialVol

Get a serial volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	12/14/98

Syntax

void SsGetSerialVol(

char *s_num*, Serial number
SndVolume **s_vol*) Pointer to volume value

Explanation

Returns the specified serial volume value to *s_vol*.

s_num can be SS_SERIAL_A for Serial A (CD input), or SS_SERIAL_B for Serial B (external digital input).

See also

[SsSetSerialVol\(\)](#)

SsGetVoiceMask

Get voices blocked from access by voice allocation system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

u_long SsGetVoiceMask(void)

Explanation

Returns a bit mask containing the voices that are blocked from access by the libsnd voice allocation system.

Return value

A value whose bits are set for each voice that is blocked. To find out if a specific voice is blocked, use the bit values SPU_xxCH(xx=0~23).

See also

[SsSetVoiceMask\(\)](#), [SsSetReservedVoice\(\)](#)

SsInit

Initialize sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsInit(*void*)

Explanation

Initializes the sound system, clearing the sound local memory.

See also

[SsInitHot\(\)](#), [SsEnd\(\)](#), [Spulnit\(\)](#) (see libspu)

SslnitHot

Initialize sound system (hot reset).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	12/14/98

Syntax

void SslnitHot(void)

Explanation

Initializes the sound system, without destroying data that has been transferred to the sound buffer. Using Exec()-related functions, when a child process wants to initialize the sound system with the sound buffer in its current state, it should call SslnitHot() instead of calling Sslnit().

See also

[Sslnit\(\)](#), Exec() (see libapi), SpulnitHot()

SsIsEos

Determine whether a song is being played.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
short SsIsEos(
short access_num,      SEQ/SEP access number
short seq_num)         SEQ number inside SEP data
```

Explanation

Determines whether or not a specified song is being played.

When using this function for SEQ data, set *seq_num* to 0; for SEP data, set it to the number of the SEQ to be played.

Return value

1 if the song is being played; 0 if the song is not being played.

SsPitchFromNote

Convert MIDI note information into a pitch playback rate.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

u_short SsPitchFromNote(
short *note*,
short *fine*,

u_char *center*,

u_char *shift*)

MIDI note number (0-127) at which the tone is keyed on.
The fine-tuning adjustment in cents, of *note*. Values range from 0-127, but are scaled to cents.
MIDI note number (0-127) at which the tone was created; the center member of the VagAtr structure.
The fine-tuning adjustment in cents, of *center*; the shift member of the VagAtr structure. Values range from 0-127, but are scaled to cents.

Explanation

Calculates a pitch value to be applied to a voice in the function SsQueueRegisters().

Return value

The calculated pitch value.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SsPlayBack

Restarts currently playing seq/sep data at beginning

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsPlayBack(
short access_num,      SEQ/SEP access number
short seq_num,         SEQ number inside SEP data
short l_count)         Song repetition count
```

Explanation

Stops the song being played, returns to the start of the song, and begins playing it again.

When using this function for SEQ data, set *seq_num* to 0. When using this function for SEP data, set the number that contains the SEQ to be played.

Specify a song repetition count in *l_count*. For infinite play repetition, specify SSPLAY_INFINITY.

See also

[SsSeqPlay\(\)](#), [SsSepPlay\(\)](#)

SsQueueKeyOn

Set voices in key on queue to be processed at next tick callback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

**void SsQueueKeyOn(
long *voices*)** Bit field containing voices to be set in the key on queue

Explanation

Hooks into the key on system of libsnd.

Set the individual bits of *voices* using the values SPU_xxCH(0-23) for the desired voices to be set in the key on queue.

This function should only be used as part of the series SsBlockVoiceAllocation() -> SsAllocateVoices()-> SsSetVoiceSettings()-> SsQueueRegisters()-> SsQueueKeyOn()-> SsQueueReverb() -> SsUnBlockVoiceAllocation().

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SsQueueRegisters

Place values in register queue to be set at next tick callback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

void SsQueueRegisters(

long *voice*,

SndRegisterAttr **SRA*)

The voice number for which the SPU registers need to be set (0-23)
Values to be sent to the SPU registers

Explanation

Queues the SndRegisterAttr to be processed at the next tick callback.

SRA members are as follows:

volume.left and *volume.right*: Any value between -0x4000 ~ 0x3fff. This is equivalent to volume mode SPU_VOICE_DIRECT (See SpsSetVoiceAttr() for details). Normally, libsnd takes only values from 0-127 and converts them to short int via the algorithm listed in the overview of libsnd. The volume calculations may be emulated or an algorithm to use fewer CPU cycles or do 3-D sound volume calculations may be substituted.

pitch: When using this function in the libsnd emulation functions series, the value set in pitch must match the value of the pitch member of the SndVoiceStats structure set in SsSetVoiceSettings(). When using this function to change pitch when the voice is currently sounding, SsPitchFromNote() may be used to calculate the pitch, or a user-defined pitch lookup table may be used.

mask: may be set by ORing the desired following list of attributes (if *mask* is set to 0, all attributes are set):

Table 14-2

Attribute	Description
SND_VOLL	left volume
SND_VOLR	right volume
SND_PITCH	pitch
SND_ADDR	waveform data start address
SND_ADSR1	adsr1 information
SND_ADSR2	adsr2 information

addr: Contains the waveform data start address. May be obtained using SsUtGetVagAddrFromTone().

adsr1: Contains adsr information for the voice. Should initially be set to the adsr1 member of VagAtr for the tone assigned to the voice.

adsr2: Contains adsr information for the voice. Should initially be set to the adsr2 member of VagAtr for the tone assigned to the voice.

This function must be used in the series: SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters()-> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation() when keying sounds on via the libsnd emulation method.

It may also be called at any time after the sound has been keyed on to change any of the members of SndRegisterAttr, provided that the function SsVoiceCheck() is called first to ensure voice integrity.

See also

SndRegisterAttr(), SndVoiceStats(), SndVolume2(), SsAllocateVoices(), SsBlockVoiceAllocation(), SsGetActualProgFromProg(), SsPitchFromNote(), SsQueueKeyOn(), SsQueueReverb(), SsSetVoiceSettings(), SsUnBlockVoiceAllocation(), SsVoiceCheck()

SsQueueReverb

Set voices in reverb queue to be processed at next tick callback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

void SsQueueReverb(

long *voices*,

Bitfield containing voices for which reverb will be changed

long *reverb*)

Bitfield containing reverb on/off data to be set in the reverb queue

Explanation

Hooks into the reverb queueing systems of libsnd, for applying reverb to sounds about to key on, altering the reverb of currently playing sounds, or applying reverb for libspu voices when both sound libraries are used together.

Set the arguments as follows:

voices

OR SPU_xxCH(0-23) for the desired voices to be set in the reverb queue.

reverb

Reverb queue bitfield.

Reverb is affected for all high bits of *voices*.

If bit = 0 Reverb is turned off for that voice

If bit = 1 Reverb is turned on for that voice

If this function is being used as part of the libsnd key-on emulation series:

SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters() -> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation(), the *mode* member of the VagAtr for the voice may be checked to determine whether reverb should affect that voice.

This function may also be used as a workaround for the reverb conflict between libspu and libsnd. Voices being used by libspu may have reverb changed using this function.

Also, this function is an ideal solution to changing reverb mode during playback of sound. In libsnd it is currently difficult to change reverb mode during playback of sound effects or MIDI music.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#), [SsVoiceCheck\(\)](#)

SsQuit

Terminate the sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsQuit(void)

Explanation

Terminates the sound system; transfer to the sound buffer is disabled. To re-enable transfer to the sound buffer, call SsInit().

SsEnd() must be called before SsQuit().

See also

[SsEnd\(\)](#), [SsStart\(\)](#), [SsSetTickMode\(\)](#), [SsSeqCalledTbyT\(\)](#)

SsSepClose

Close SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSepClose(  
short sep_access_num)    SEP access number
```

Explanation

Closes SEP data with *sep_access_num* that is no longer needed.

All SEQ data stored in the closed SEP becomes inaccessible. Before executing this function, use SsSepStop() with the applicable SEP.

See also

[SsSepOpen\(\)](#)

SsSepOpen

Open SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
short SsSepOpen(
  u_long *addr,           Pointer to starting address of SEP data within the main memory
  short vab_id,           VAB id
  short seq_num)         Number of SEQs contained in SEP
```

Explanation

Analyzes the SEP data located in the main memory, and returns a SEP access number. Up to 32 pieces of SEP data can be opened simultaneously when combined with the number of open SEQ data.

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your *s_max* from *SsSetTableSize()* is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by *s_max*.

Return value

SEP access number: an internal SEP data management table number that has the same characteristics as the SEQ access number.

See also

[SsSepClose\(\)](#)

SsSepOpenJ

Open SEP data (function table version).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	12/14/98

Syntax

```
short SsSepOpenJ(
  u_long *addr,           Pointer to starting address of SEP data within the main memory
  short vab_id,           VAB id
  short seq_num)          Number of SEQs contained in SEP
```

Explanation

Equivalent to SsSepOpen() if all the low-level MIDI functions were registered. In addition to the SsSepOpen() capability, this function enables a programmer to control functions to be registered to the table and thus improve code efficiency by not calling unnecessary low-level functions.

Failure to properly register all necessary low-level MIDI functions will result in a bus error.

Before calling this function, you must have confirmed with SsVabTransCompleted() that the VAB data from the *vab_id* has completed being transferred to the sound buffer.

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your *s_max* from SsSetTableSize() is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by *s_max*.

Return value

SEQ Access Number: Used in the SEQ data access function, being the inner SEQ data control table number.

See also

[SsSepOpen\(\)](#), [_SsFCALL\(\)](#), [dmy_Ss...\(\)](#)

SsSepPause

Pause the reading of SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSepPause(  
short sep_access_num,    SEP access number  
short seq_num)           SEQ number inside SEP data
```

Explanation

Pauses the reading (playing) of the *seq_num* SEQ data of SEP data possessing *sep_access_num*.

See also

[SsSepReplay\(\)](#)

SsSepPlay

Read (play) SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSepPlay(
short *sep_access_num*, SEP access number
short *seq_num*, SEP data SEQ number
char *play_mode*, Play mode
short *l_count*) Song repetition count

Explanation

Begins to read (play) SEQ data specified by the SEP data *seq_num* specified by *sep_access_num*, if *play_mode* = SSPLAY_PLAY. If *play_mode* = SSPLAY_PAUSE, makes a pause state. For infinite play repetition, specify SSPLAY_INFINITY in *l_count*.

Example:

```
sep1 = SsSepOpen (addr, vab_id, 4); /* Open SEP data containing four pieces
of SEQ data */
SsSepPlay (sep1, 2, SSPLAY_PLAY, 2); /* Immediately play 3rd SEQ data of
opened SEP data twice */
```

See also

[SsSepStop\(\)](#)

SsSepReplay

Resume (replay) reading SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSepReplay(
short sep_access_num,    SEP access number
short seq_num)           SEQ number inside SEP data
```

Explanation

Resumes reading the *seq_num*-th SEQ data of SEP data with *sep_access_num*, that was paused by SsSepPause().

See also

[SsSepPause\(\)](#)

SsSepSetAccelerando

Accelerate the tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSepSetAccelerando(

short <i>sep_access_num</i> ,	SEP access number
short <i>seq_num</i> ,	SEQ number inside SEP data
long <i>tempo</i> ,	Desired song tempo
long <i>v_time</i>)	Time (in ticks)

Explanation

Increases the tempo of the *seq_num*-th SEQ data of SEP data with *sep_access_num* up to *tempo* within *v_time*.

If *tempo* is smaller (slower) than the current tempo, this function acts the same as `SsSepSetRitardando()`.

See also

[SsSepSetRitardando\(\)](#)

SsSepSetCrescendo

Crescendo (valid for individual SEQ in SEP).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSepSetCrescendo(
short sep_access_num,    SEP access number
short seq_num,           SEQ number inside SEP data
short vol,               Volume value (0-127)
long v_time)             Time (in tick units)
```

Explanation

Raises the main volume of the *seq_num*-th SEQ data of SEP data with *sep_access_num* by *vol* within *v_time* (or to the maximum value). It has no effect if the volume of each voice is at the maximum or if *vol* is a negative number. It is recommended that *v_time* be set to an integer multiple of *vol*.

See also

[SsSepSetDecrescendo\(\)](#)

SsSepSetDecrescendo

Decrescendo (valid for individual SEQ in SEP).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSepSetDecrescendo(

short *sep_access_num*, SEP access number
short *seq_num*, Number inside SEP data
short *vol*, Volume value (0-127)
long *v_time*) Time (in tick units)

Explanation

Lowens the main volume of the *seq_num*-th SEQ data of SEP data with *sep_access_num* by *vol* within *v_time* (or to the minimum value). This function has no effect if the volume of each voice is at the minimum or if *vol* is a negative number. It is recommended that *v_time* be set to an integer multiple of *vol*.

See also

[SsSepSetCrescendo\(\)](#)

SsSepSetRitardando

Slow the tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSepSetRitardando(

short *sep_access_num*, SEP access number
short *seq_num*, SEQ number inside SEP data
long *tempo*, Desired song tempo
long *v_time*) Time (in tick units)

Explanation

Slows the tempo of the *seq_num* SEQ data of SEP data possessing *sep_access_num* down to tempo within *v_time*. If *tempo* is larger (faster) than the current tempo, this function acts the same as SsSepSetAccelerando().

See also

[SsSepSetAccelerando\(\)](#)

SsSepSetVol

Set SEP volume (valid for individual SEQ in SEP).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSepSetVol(  
short sep_access_num, : SEP access number  
short seq_num,        : SEQ number inside SEP data  
short voll,           : L channel main volume value  
short volr)           : R channel main volume value
```

Explanation

Sets the L and R channels for the main volume of the *seq_num*-th SEQ data of SEP data with *sep_access_num* to specified values (between 0 and 127).

SsSepStop

Stop reading SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSepStop(
short sep_access_num,    SEP access number
short seq_num)           SEQ number inside SEP data
```

Explanation

Stops reading (playing) the *seq_num*-th SEQ data of SEP data with *sep_access_num*.

See also

[SsSepPlay\(\)](#)

SsSeqCalledTbyT

Interpret SEQ/SEP data and carry out playback.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSeqCalledTbyT(void)

Explanation

At each Tick this function is called; it interprets SEQ/SEP data and carries out playback. The tick rate is set by `SsSetTickMode()`, but this merely regulates the internal sound system, without depending either on the speed or resolution determined by SEQ/SEP data.

When `SsSetTickMode()` is called with *tick_mode* `SS_NOTICK`, this function must be called by the program (usually with vertical sync timing). Otherwise, the sound processing code automatically calls this function at the given resolution.

See also

[SsSetTickMode\(\)](#)

SsSeqClose

Close SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqClose(  
short seq_access_num)    SEQ access number
```

Explanation

Closes SEQ data with an un-needed *seq_access_num*.

Before executing this function, use SsSeqStop() with the applicable SEQ.

See also

[SsSeqOpen\(\)](#)

SsSeqGetVol

Get SEQ volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqGetVol(
short access_num,      SEQ/SEP access number
short seq_num,         SEQ number of SEP data
short *voll,           L volume of SEQ data
short *volr)           R volume of SEQ data
```

Explanation

Returns current left and right SEQ volume to *voll* and *volr*. Set *seq_num* at 0 for SEQ data, and set it at appropriate SEQ number for SEP data. The volume value set by *SsSepSetVol()* can be obtained.

See also

[SsSeqSetVol\(\)](#), [SsSepSetVol\(\)](#)

SsSeqOpen

Open SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
short SsSeqOpen(
u_long *addr,      Pointer to start address of SEQ data in the main storage
short vab_id)      VAB id
```

Explanation

Designates an SEQ number for the SEQ data to allow playback.

Before calling this function, you must have confirmed with `SsVabTransCompleted()` that the VAB data from the *vab_id* has completed being transferred to the sound buffer.

For Library Versions 4.0 and earlier:

- 1) Do not call this function from inside a callback;
- 2) If your *s_max* from `SsSetTableSize()` is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by *s_max*.

Return value

SEQ access number. This value is passed to other SEQ routines such as `SsSeqPlay()`.

If you try to open more than 32 SEP data (combined with SEQ data) at the same time, -1 is returned.

See also

[SsSeqClose\(\)](#)

SsSeqOpenJ

Open SEQ data (function table version).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.6	12/14/98

Syntax

```
short SsSeqOpenJ(
  u_long *addr,      Pointer to start address of SEQ data in the main storage
  short vab_id)      VAB id
```

Explanation

Equivalent to SsSeqOpen() if all the low-level functions were registered. In addition to the SsSeqOpen() capability, this function enables a programmer to control functions to be registered to the table and thus improve code efficiency by not calling unnecessary low-level functions.

Failure to properly register all necessary low-level MIDI functions will result in a bus error.

Before calling this function, you must have confirmed with SsVabTransCompleted() that the VAB data from the *vab_id* has completed being transferred to the sound buffer.

For Library Versions 4.0 and earlier:

- 1) Do not call this from inside a callback;
- 2) If your *s_max* from SsSetTableSize() is less than 32, you must keep track of your open SEQs/SEPs so as not to exceed the limit set by *s_max*.

Return value

SEQ access number. This value is passed to other SEQ routines such as SsSeqPlay().

See also

[SsSeqOpen\(\)](#)

SsSeqPause

Pause SEQ data reading.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqPause(
short seq_access_num)  SEQ access number
```

Explanation

Stops reading (playing) SEQ data with *seq_access_num*.

See also

[SsSeqReplay\(\)](#)

SsSeqPlay

Read (play) SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqPlay(
short seq_access_num,    SEQ access number
char play_mode,          Performance mode
short l_count)           Number of repeats of the music
```

Explanation

Selects either immediate SEQ data reading (*play_mode* = SSPLAY_PLAY) or sets a pause state at the start of SEQ data (*play_mode* = SSPLAY_PAUSE). Specify the number of times to repeat the music by *l_count*, using SSPLAY_INFINITY for unlimited play.

See also

[SsSeqPause\(\)](#)

SsSeqPlayPtoP

Read SEQ data (play interval).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.2	12/14/98

Syntax

```
void SsSeqPlayPtoP(
short access_num,      SEQ/SEP access number
short seq_num,         SEQ number within SEP (0 if music data is SEQ)
u_char *start_point,   Load (play) start point
u_char *end_point,     Load (play) end point
char play_mode,        Performance mode
short l_count)         Loop count
```

Explanation

Loads (plays) the data interval specified by *start_point* and *end_point*. These values are obtained from *SsGetCurrentPoint()*.

If *play_mode* = SSPLAY_PLAY, SEQ data is read (played) immediately. If *play_mode* = SSPLAY_PAUSE, a pause is entered at the start of the SEQ data (the start of the song).

l_count specifies the number of times the song is to be looped. Use SSPLAY_INFINITY for an infinite loop.

See also

[SsSeqPlay\(\)](#), [SsSepPlay\(\)](#), [SsGetCurrentPoint\(\)](#)

SsSeqReplay

Resume SEQ data reading (Replay) .

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

**void SsSeqReplay(
short *seq_access_num*)** SEQ access number

Explanation

Resumes reading SEQ data with *seq_access_num* stopped by SsSeqPause().

See also

[SsSeqPause\(\)](#)

SsSeqSetAccelerando

Quicken tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqSetAccelerando(
short seq_access_num,    SEQ access number
long tempo,              Desired music tempo
long v_time)             Time (in ticks)
```

Explanation

Quickens the SEQ data with *seq_access_num* to the *tempo* resolution in *v_time*. If the specified resolution is smaller (slower) than the current resolution, the effect is the same as *SsSeqSetRitardando()*.

See also

[SsSeqSetRitardando\(\)](#)

SsSeqSetCrescendo

Crescendo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSeqSetCrescendo(

short *seq_access_num*, SEQ access number

short *vol*, Volume value (0-127)

long *v_time*) Time (in ticks)

Explanation

Increases the main volume of SEQ data with *seq_access_num* by the *vol* value in *v_time*. If the voice volume is at the maximum (127), or if *vol* is a negative number, the function has no effect. It is recommended that *v_time* be set to an integer multiple of *vol*.

See also

[SsSeqSetDecrescendo\(\)](#)

SsSeqSetDecrescendo

Decrescendo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqSetDecrescendo(
short seq_access_num,      SEQ access number
short vol,                 Volume value (0-127)
long v_time)               Time (in ticks)
```

Explanation

Lowers main volume of SEQ data with *seq_access_num* by the *vol* value in *v_time*. If each voice volume is the minimum value (0), or if *vol* is a negative number, there is no effect. It is recommended that *v_time* be set to an integer multiple of *vol*.

See also

[SsSeqSetCrescendo\(\)](#)

SsSeqSetNext

Specify SEQ data to play next after a given SEQ.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSeqSetNext(

short *seq_access_num1*, SEQ access number

short *seq_access_num2*) SEQ access number of next SEQ to play

Explanation

Specifies that whenever SEQ data represented by *seq_access_num1* plays, the SEQ data represented by *seq_access_num2* should play next.

See also

[SsSetNext\(\)](#)

SsSeqSetRitardando

Slow tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqSetRitardando(
short seq_access_num,      SEQ access number
long tempo,                Desired music tempo
long v_time)               Time (in ticks)
```

Explanation

Slows the SEQ data with *seq_access_num* to the tempo resolution *in v_time*. If the specified resolution is larger (faster) than the current resolution, the function is the same as *SsSeqSetAccelerando()*.

See also

[SsSeqSetAccelerando\(\)](#)

SsSeqSetVol

Set SEQ volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSeqSetVol(  
  short seq_access_num,  SEQ access number  
  short voll,            Left channel's main volume value  
  short volr)            Right channel's main volume value
```

Explanation

Sets the main volume of music with *seq_access_num* to values specified for *voll* and *volr* (from 0 to 127).

SsSeqSkip

Skip SEQ data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.2	12/14/98

Syntax

```
int SsSeqSkip(
short access_num,      SEQ/SEP access number
short seq_num,         SEQ number within SEP (0 if music data is SEQ)
char unit,             Skip unit
short count)           Skip amount
```

Explanation

Moves the playback pointer of the song data represented by *access_num* and *seq_num* forward *count* times, in units specified by *unit*.

Table 14-3

Unit	Skip unit
SSSKIP_TICK	TICK unit
SSSKIP_NOTE4	Quarter note unit
SSSKIP_NOTE8	One-eighth note unit
SSSKIP_BAR	Measure unit

Return value

0 if the function was successful; -1 if the operation failed

SsSeqStop

Terminate SEQ data reading.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
void SsSeqStop(  
short seq_access_num)   SEQ access number
```

Explanation

Terminates playing of the SEQ data with *seq_access_num*.

See also

[SsSeqPlay\(\)](#)

SsSetAutoKeyOffMode

Set automatic KeyOff mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSetAutoKeyOffMode(
short mode)
```

0: Automatically keys off
1: Does not key off until a KeyOff request comes in

Explanation

Sets the automatic KeyOff mode. The default is the automatic KeyOff mode. If the envelopes for a specific voice for the past 16 interrupts contain all 0's, the automatic KeyOff mode assumes that waveform playback has been automatically terminated, and uses the voice for other waveform playback.

SsSetCurrentPoint

Set data address in SEQ/SEP.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.2	12/14/98

Syntax

```
int SsSetCurrentPoint(
short acn,           SEP access number
short trn,           SEQ number within SEP (0 if music data is SEQ)
u_char *point)      SEQ/SEP data address
```

Explanation

Sets the data address obtained from SsGetCurrentPoint() in SEQ/SEP.

Return value

0 if the function was successful

-1 if the operation failed

See also

[SsGetCurrentPoint\(\)](#)

SsSetLoop

Set song repetition count.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSetLoop(
short access_num,      SEQ/SEP access number
short seq_num,         SEQ number inside SEP data
short l_count)         Song repetition count
```

Explanation

Sets a song repetition count. This function is useful for changing the song repetition count set in SsSeqPlay. After this function is called, the current song repetition count is reset, and the song is played for the number of times set by the new count.

When using this function for SEQ data, set 0 in *seq_num*; when using this function for SEP data, set the number that contains the SEQ to be played.

SsSetMarkCallback

Register a function to be called when a mark is detected.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
typedef void
(*SsSeqMarkCallbackProc)
(short, short, short);
```

```
void SsSetMarkCallback(
short access_num,      SEQ/SEP access number
short seq_num,         SEQ number inside SEP data
SsMarkCallbackProc proc) Callback function to be called when Mark is detected
```

Explanation

proc specifies a callback function to be called when a mark is detected inside a song identified by *access_num*. If *proc* is 0, any previous callback is cleared. Only one callback function can be registered at a time. The arguments passed to the callback function are, in order:

- SEQ/SEP number
- SEQ number inside SEP data (0 when using SEQ)
- The data2 value following the callback marker in the song data is passed to the third argument.

Sample:

```
/* Callback function-definition*/
SsMarkCallbackProc proc (short ac_no, short tr_no, short
data);
:
/* Opens SEQ */
short seq_a_num = SsSeqOpen (addr, vab_id);
/* Sets Callback function */
SsSetMarkCallback (seq_a_num, 0, (SsMarkCallbackProc) proc);
```

SsSetMono

Set monaural mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsSetMono(void)

Explanation

Sets the output to monaural mode. Forces all libsnd keyed-on voices to have equivalent left and right volumes. Stereo mode is the system default mode.

See also

[SsSetStereo\(\)](#)

SsSetMute

Sets Muting.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
void SsSetMute(  
char mode)           Setting mode
```

Explanation

If *mode* is SS_MUTE_ON, the sound system is muted. If *mode* is SS_MUTE_OFF, the sound system is unmuted.

SsSetMVol

Set main volume value.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsSetMVol(

short *voll*, L channel volume value

short *volr*) R channel volume value

Explanation

Sets the master volume for the sound system to *voll* and *volr* (values range from 0 to 127).

You must call this function before playing sequence (SEQ, SEP) data.

See also

[SsGetMVol\(\)](#)

SsSetNext

Set the next SEQ/SEP data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.2	12/14/98

Syntax

void SsSetNext(

short *ac_no1*,

SEP/SEQ access number

short *tr_no1*,

SEQ number in SEP (If the score data is SEQ, *tr_no1* is 0.)

short *ac_no2*,

SEP/SEQ access number

short *tr_no2*)

SEQ number in SEP (If the score data is SEQ, *tr_no2* is 0.)

Explanation

Sets the score data with SEP/SEQ access numbers (*ac_no2*, *tr_no2*) to be played after SEP/SEQ data (*ac_no1*, *tr_no1*).

The next score data is played automatically after the previous score finishes playing.

See also

[SsSeqSetNext\(\)](#)

SsSetReservedVoice

Declare the number of voices to be allocated by libsnd.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
char SsSetReservedVoice(
char voices)           Voice count
```

Explanation

Declares the number of voices that the libsnd voice allocation management system has access to. Other voices can be keyed on in libspu or via SsUtKeyOnV().

Voice numbers are reserved from the lower end (from 0). For example, if *voices* = 20, then:

- Voices 0-19 are used for allocation by libsnd.
- Voices 20-23 are available for libspu.

Should only be called once, before start of sound processing (SsStart/SsStart2()).

Return value

Returns the set voice count if successful. Returns -1 if unsuccessful.

SsSetRVol

Set reverb volume values.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsSetRVol(

short *voll*, L channel's volume value

short *volr*) R channel's volume value

Explanation

Sets the reverb volume for left and right channels. The value ranges from 0 to 127.

See also

[SsGetRVol\(\)](#)

SsSetSerialAttr

Set a serial attribute.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
void SsSetSerialAttr(
char s_num,           Serial number
char attr,            Attribute value
char mode)           Setting mode
```

Explanation

Sets a serial attribute.

- *s_num*: SS_SERIAL_A = Serial A (CD input). SS_SERIAL_B = Serial input line B (external digital input)
- *attr*: SS_MIX = Mixing. SS_REV = Reverb
- *mode*: SS_SON = attr on. SS_SOFF = attr off

See also

[SsGetSerialAttr\(\)](#)

SsSetSerialVol

Set serial volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsSetSerialVol(

char *s_num*, Serial number
short *voll*, L channel volume value
short *volr*) R channel volume value

Explanation

Sets the serial volume in *voll*, *volr*. The volume values may be between 0-127.

s_num: SS_SERIAL_A = Serial A (CD input). SS_SERIAL_B = Serial input line B (external digital input)

See also

[SsGetSerialVol\(\)](#)

SsSetStereo

Set stereo mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsSetStereo(void)

Explanation

Sets the output to stereo mode. The sound system default output is stereo.

See also

[SsSetMono\(\)](#)

SsSetTableSize

Specify the area of a SEQ/SEP data attribute table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsSetTableSize(
char **table*, Pointer to SEQ/SEP data attribute table area variable
short *s_max*, Maximum times to open SEQ/SEP data (up to 32)
short *t_max*) Number of SEQ included in SEP

Explanation

Specifies the area of a SEQ/SEP data attribute table. The library uses this area to analyze SEQ/SEP data, then saves it and plays it back.

s_max specifies the maximum number of times SEQ/SEP data may be opened. The upper limit is 32. Once *s_max* is reached, unused SEQ/SEP data must be closed with SsSeqClose()/SsSepClose() before more data can be opened. *t_max* specifies the number of SEQ included in the largest SEP data. Set *t_max* to 1 to handle only SEQ data and not use SEP data. The upper limit of *t_max* is 16.

You must preserve the area for the table by using global variables or functions like malloc() (auto variables cannot be used).

Use the following to find the size:

$$(SS_SEQ_TABSIZ \times s_max \times t_max)$$

where the constant SS_SEQ_TABSIZ is declared in libsnd.h. (note that its value may vary from version to version).

SsSetTableSize() should be called immediately after SsInit(). Both functions should be called only once; what happens when multiple calls are made is unclear.

See also

SsInit(), SsInitHot(), SsSeqOpen(), SsSepOpen()

SsSetTempo

Set a tempo.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsSetTempo(
short access_num,    SEQ/SEP access number
short seq_num,       SEQ number inside SEP data
short tempo)         Song tempo
```

Explanation

Sets a tempo. This function is useful for changing the tempo set in SsSeqPlay().

After this function is called, the current tempo is changed to the new tempo specified for playing songs.

When using this function for SEQ data, set 0 in *seq_num*; when using this function for SEP data, set the number that contains the SEQ to be played.

SsSetTickCallback

Set the TickCallBack function called with every TICK.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
int SsSetTickCallback(
void (*cb)())           Pointer to TickCallBack function called with every Tick
```

Explanation

Defines *cb* as the TickCallBack function called every tick. It is called only when SS_NOTICK has not been set by SsSetTickMode(), after SsStart() or SsStart2() have been called.

When this function isn't used to set the TickCallBack function, the default is SsSeqCalledTbyT().

Return value

Previously-set TickCallback function.

See also

[SsSetTickMode\(\)](#), [SsStart\(\)](#), [SsStart2\(\)](#), [SsSeqCalledTbyT\(\)](#)

SsSetTickMode

Set tick mode.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
void SsSetTickMode(
long tick_mode)           Tick mode (see table for values)
```

Explanation

Sets the resolution of a tick. Call this function only once before calling SsSeqOpen(), SsSepOpen() or SsStart() for the first time. When it is called multiple times, correct operation cannot be guaranteed.

The tick mode does not depend on the speed or resolution specified by SEQ/SEP data, and merely specifies the resolution inside the sound system.

The effects of SS_TICK50, SS_TICK60, and SS_TICKVSYNC differ according to the specification of SetVideoMode() (see the individual entries below).

tick_mode may be specified with the following values:

Table 14-4

<i>tick_mode</i>	Tick setting: SEQ file played at
SS_TICK50	1/50 second
SS_TICK60	1/60 second
SS_TICKVSYNC	VSync Resolution (1/50 PAL, 1/60 NTSC)
SS_TICK120	1/120 second
SS_TICK240	1/240 second
SS_NOTICK	1/60 second*
Any resolution (60-240)	1/ <i>tick_mode</i> seconds
Any resolution SS_NOTICK	1/ <i>tick_mode</i> seconds*

* SsSeqCalledTbyT() is called automatically every tick, except when *tick_mode* is SS_NOTICK or (any resolution | SS_NOTICK). In those cases, the program must call SsSeqCalledTbyT() at the specified timing.

“Any resolution” means that you specify a value between 60-240, and the resolution is 1/*tick_mode*. Example: *tick_mode* = 65 | SS_NOTICK sets up a resolution of 1/65th second.

The OS Root Counter RCntCNT3 is used to generate VSYNC timing. Therefore, if you use SS_TICK50 with MODE_PAL (specified in SetVideoMode()), or SS_TICK60 with MODE_NTSC, or SS_TICKVSYNC, you shouldn't use RCntCNT3 for any other timing resolution.

The OS Root Counter RCntCNT2 is used for all other tick modes. Therefore, you shouldn't use RCntCNT2 for any other timing resolution.

See also

SsStart(), SsSeqCalledTbyT(), SetVideoMode() (see libetc), SsSetTickCallback()

SsSetVoiceMask

Block voice allocation system from using the specified voices.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

```
void SsSetVoiceMask(  
u_long s_voice)      voices that libsnd allocation system will not have access to. Specify the  
                        voices with the bits SPU_0CH..SPU_23CH.
```

Explanation

Blocks the voices specified in the bit mask *s_voice* from being accessed by the libsnd voice allocation system. These voices are then available for use via libspu calls or SsUtKeyOnV() calls. This function differs from SsSetReservedVoice() in that a non-contiguous block of voices may be specified. For example, SsSetVoiceMask(6) gives the libsnd voice allocation system access to 22 voices - voices 0 and 3-23, while SsSetReservedVoice(22) gives it access to voices 0-21. The two functions may be used together, so that individual voices within the limits set by SsSetReservedVoice() may be blocked, but an easier method is to make one call to SsSetVoiceMask().

SsAllocateVoices() doesn't include code to block these voices from the libsnd voice allocation system.

See also

[SsGetVoiceMask\(\)](#), [SsSetReservedVoice\(\)](#)

SsSetVoiceSettings

Set internal libsnd variables for a voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

```
void SsSetVoiceSettings(
long voice,           Voice number (0-23)
SndVoiceStats *Snd_v_attr)  Voice attributes to be set
```

Explanation

Hooks into the internal libsnd variables before key on of sound so that libsnd functions such as SsUtPitchBend() can be used to alter the voice after key on.

The voice attributes *Snd_v_attr* must be set as follows:

- *vagId*: VAG number which tone to be keyed on points to. Can be obtained by using SsUtGetVagAtr().
- *vabId*: VAB in which information for tone to be keyed on resides.
- *pitch*: Original playback rate of tone. May be obtained by using SsFindPitch() or by creating a user-defined pitch lookup table.
- *vol*: Maximum volume at which tone will be played back. Choose the greater value between left and right channel. Negative values are not valid (*vol* is invalid when used with 3D sound), and values range from 0-127.
- *pan*: Relative pan value at which tone will be keyed on at. Not valid with 3D sound. Use the formula $pan = volr * 64 / voll$ if $volr > voll$ and $pan = \text{max volume (either } 0x3ff \text{ or } 127) - voll * 64 / volr$ if $voll > volr$. If $voll = volr$, $pan = 64$.
- *note*: Note at which tone will be keyed on.
- *tone*: Tone number to be keyed on. Determined by comparing the desired note with the *min* and *max* members of VagAtr for all tones within the specified program and *vabId*.
- *prog_num*: The program number within which the tone information resides.
- *prog_actual*: The “real” program number within which the tone information resides. The “real” number is only incremented by valid programs (programs containing one or more tones) and so may differ from the program number. Example: In a VAB with valid programs 0-10 and 100-127, the *prog_num* of program 127 = 127, while the *prog_actual* of program 127 = 38. This number is used to calculate the offset of tone information in the VAB header and may be obtained using SsGetActualProgFromProg().

This function should only be used as part of the series: SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters() -> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation()

See also

SndRegisterAttr(), SndVoiceStats(), SndVolume2(), SsAllocateVoices(), SsBlockVoiceAllocation(), SsGetActualProgFromProg(), SsPitchFromNote(), SsQueueKeyOn(), SsQueueRegisters(), SsQueueReverb(), SsUnBlockVoiceAllocation(), SsVoiceCheck()

SsStart

Start the sound system.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

void SsStart(void)

Explanation

Starts the sound system.

When `SsSetTickMode()` is used to set a mode that calls `SsSeqCalledTbyT()` automatically, this function causes `SsSeqCalledTbyT()` to be called each tick.

See also

[SsEnd\(\)](#), [SsSetTickMode\(\)](#), [SsSeqCalledTbyT\(\)](#)

SsStart2

Start the sound system (VSyncCallback version).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	2/24/99

Syntax

void SsStart2(void)

Explanation

Starts the sound system.

When `SsSetTickMode()` is used to set a mode that calls `SsSeqCalledTbyT()` automatically, this function causes `SsSeqCalledTbyT()` to be called each tick.

`SsStart2()` must be used when the tick mode is equal to the TV's sync rate (e.g. `SS_TICK60` on NTSC or `SS_TICK50` on PAL). It may also be used with other modes.

See also

[SsStart\(\)](#), [SsEnd\(\)](#), [SsSetTickMode\(\)](#), [SsSeqCalledTbyT\(\)](#)

SsUnBlockVoiceAllocation

Release voice allocation system used by SsUtKeyOn(), SsUtKeyOnV(), and MIDI key on commands.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	4.1	12/14/98

Syntax

char SsUnBlockVoiceAllocation(void)

Explanation

Releases the voice allocation system used by SsUtKeyOn(), SsUtKeyOnV(), SsAllocateVoices(), and MIDI key on commands. Must follow every call to SsBlockVoiceAllocation().

This function should only be used as part of the series: SsBlockVoiceAllocation() -> SsAllocateVoices() -> SsSetVoiceSettings() -> SsQueueRegisters() -> SsQueueKeyOn() -> SsQueueReverb() -> SsUnBlockVoiceAllocation().

Return value

1 if successful; -1 if voice allocation system was not currently blocked.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsVoiceCheck\(\)](#)

SsUtAllKeyOff

Key off all voices.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsUtAllKeyOff(
short mode)           Always 0
```

Explanation

Keys off all voices used by the libsnd voice allocation system, as set by SsSetReservedVoice() and SsSetVoiceMask().

SsUtAutoPan

Automatically change panning.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtAutoPan(

short *vc*, Voice number (0-23)
short *start_pan*, Panning change start value (0-127)
short *end_pan*, Panning change end value (0-127)
short *delta_time*) Change time in ticks (0-10800)

Explanation

Linearly changes the panning from *start_pan* to *end_pan* at *delta_time* (in ticks) for voice *vc*.

Return value

0 if successful; -1 if unsuccessful.

See also

[SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtAutoVol

Automatically change voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtAutoVol(

short *vc*, Voice number (0-23)
short *start_vol*, Volume change start value (0-127)
short *end_vol*, Volume change end value (0-127)
short *delta_time*) Change time in ticks (0-10800)

Explanation

Linearly changes the volume from *start_vol* to *end_vol* at *delta_time* (in ticks) for voice *vc*.

Return value

0 if successful; -1 if unsuccessful.

See also

[SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtChangeADSR

Change ADSR.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtChangeADSR(

short *vc*, Voice number (0-23)
short *vabld*, VAB number (0-31) returned by SsVabOpenHead()
short *prog*, Program number (0-127)
short *old_note*, Previous pitch specification in half-tone units (note number)(0-127)
u_short *adsr1*, ADSR1
u_short *adsr2*) ADSR2

Explanation

Changes the ADSR of the key on voice using SsUtKeyOn().

Return value

0 if successful; -1 if unsuccessful.

See also

[SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#), [SsVabOpenHead\(\)](#)

SsUtChangePitch

Change the pitch.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtChangePitch(

short *voice*,

short *vabId*,

short *prog*,

short *old_note*,

short *old_fine*,

short *new_note*,

short *new_fine*)

Voice number (0-23)

VAB number (0-31) returned by SsVabOpenHead()

Program number (0-127)

Previous pitch specification in semitones (note number) (0-127)

Previous fine pitch specification (100/127 cents) (0-127)

New pitch specification in semitones (note number) (0-127)

New fine pitch specification (100/127 cents) (0-127)

Explanation

Changes the pitch of the voice.

Return value

0 if successful; -1 if unsuccessful.

See also

[SsUtPitchBend\(\)](#), [SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtFlush

Execute remaining queued KeyOn/KeyOff requests.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsUtFlush(void)

Explanation

Executes the remaining KeyOn/KeyOff requests that have been queued.

Normally, flushing is performed by an automatic Sound Library interrupt (when the tick mode isn't SS_NOTICK) or by a clear call of SsSeqCalledTbyT (when the tick mode is SS_NOTICK). This function can also be used for flushing. It's necessary to wait an interval of at least 1/44100 sec between consecutive calls to this function and/or SsSeqCalledTbyT().

SsUtGetDetVVol

Get detailed value of voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtGetDetVVol(

short *vc*, Voice number (0-23)

short **detvoll*, Pointer to detailed volume, left (0-16383)

short **detvolr*) Pointer to detailed volume, right (0-16383)

Explanation

Returns the detailed value of the voice volume.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtSetDetVVol\(\)](#), [SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtGetProgAtr

Get a program attribute table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtGetProgAtr(
 short *vabId*, VAB number (0-31) returned by SsVabOpenHead()
 short *progNum*, Program number (0-127)
 ProgAtr **progatrptr*) Pointer to program attribute table

Explanation

Specifies a VAB number and a program number, and returns the VAB attribute table to *progatrptr*.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtSetProgAtr\(\)](#), [ProgAtr\(\)](#)

SsUtGetReverbType

Get a reverb type.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short

SsUtGetReverbType(void)

Explanation

Obtains the current reverb type value.

Return value

Current reverb type value.

See also

[SsUtSetReverbType\(\)](#).

SsUtGetVabHdr

Get VAB attribute header.

Library	Header File	Introduced	Documentation Date
libsnd.lib	libsnd.h	2.x	12/14/98

Syntax

short SsUtGetVabHdr(
short *vabId*, VAB number (0-31) returned by SsVabOpenHead()
VabHdr **vabhdrptr*) Pointer to VAB attribute header

Explanation

Returns the VAB attribute header to *vabhdrptr* of the VAB specified by *vabId*.

Return value

0 if successful, -1 if unsuccessful.

See also

[VabHdr\(\)](#)

SsUtGetVagAddr

Get an SPU buffer address stored by VAG.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.1	12/14/98

Syntax

```
long SsUtGetVagAddr(
short vabId,      VAB data id
short vagId)     VAG data id
```

Explanation

Given VAB id (0-15) and VAG id (1-254), this function returns a 32-bit SPU buffer address (as bytes) stored by VAG.

Return value

An SPU buffer address stored by VAG.

See also

[SsVabOpenHead\(\)](#)

SsUtGetVagAddrFromTone

Get SPU buffer address where VAG data is stored.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.3	12/14/98

Syntax

```
u_long SsUtGetVagAddrFromTone(  
short vabid,           VAB id  
short progid,         Program number  
short toneid)          Tone number
```

Explanation

This function returns the address in the sound buffer where the VAG wave form data with the specified VAB id, program number, and tone number are transferred.

Return value

Address in the sound buffer. If it fails, it returns -1.

SsUtGetVagAtr

Get tone attribute table (VagAtr).

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
short SsUtGetVagAtr(
short vabId,           VAB number (0-31) returned by SsVabOpenHead()
short progNum,       Program number (0-127)
short toneNum,       Tone number (0-15)
VagAtr *vagatrptr)   Pointer to tone attribute table
```

Explanation

Specifies a VAB number, a program number, and a tone number, and returns a tone attribute table to *vagatrptr*.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtSetVagAtr\(\)](#), [VagAtr\(\)](#)

SsUtGetVBaddrInSB

Get address in sound buffer to which VAB data has been transferred.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
u_long SsUtGetVBaddrInSB(  
short vabid)           VAB id
```

Explanation

Returns the address inside the sound buffer to which the start of the VAB data specified by *vabid* has been transferred.

Return value

Address inside the sound buffer. Returns -1 if unsuccessful.

SsUtGetVVol

Get voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtGetVVol(

short *vc*, Voice number (0-23)
short **voll*, Pointer to volume, left (0-127)
short **volr*) Pointer to volume, right (0-127)

Explanation

Returns a volume value for a voice.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtSetVVol\(\)](#), [SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtKeyOff

Key off voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsUtKeyOff(

short *voice*,

short *vabld*,

short *prog*,

short *tone*,

short *note*)

Voice number (0-23) access number

VAB number (0-31) returned by SsVabOpenHead()

Program number (0-127)

Tone number (0-15)

Pitch specification in half-tone units (note number) (0-127)

Explanation

Keys off the voice.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtKeyOffV

Key off a voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

**short SsUtKeyOffV(
short *voice*)** Voice number (0-23)

Explanation

Keys off the voice specified by *voice* (0-23).

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtKeyOnV\(\)](#)

SsUtKeyOn

Key on a voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsUtKeyOn(
short *vabId*, VAB number (0-31) returned by SsVabOpenHead()
short *prog*, Program number (0-127)
short *tone*, Tone number (0-15)
short *note*, Pitch specification in semitones (note number) (0-127)
short *fine*, Detailed pitch specification (100/127 cents) (0-127)
short *voll*, Volume, left (0-127)
short *volr*) Volume, right (0-127)

Explanation

Keys on the voice specified by the VAB number, the program number (0-127), and the tone number (0-15) at the specified pitch and volume, and returns the allocated voice number.

Return value

The voice number (0-23) used for KeyOn. Returns -1 if unsuccessful.

See also

[SsUtKeyOff\(\)](#)

SsUtKeyOnV

Key on a voice.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsUtKeyOnV(

short *voice*,

Voice number (0-23)

short *vabld*,

VAB number (0-31) returned by SsVabOpenHead()

short *prog*,

Program number (0-127)

short *tone*,

Tone number (0-15)

short *note*,

Pitch specification in semitones (note number) (0-127)

short *fine*,

Detailed pitch specification (100/127 cents) (0-127)

short *voll*,

Volume, left (0-127)

short *volr*)

Volume, right (0-127)

Explanation

Keys on the voice specified by the voice number (0-23), the VAB number, the program number (0-127), and the tone number (0-15) at the specified pitch and volume, and returns the allocated voice number.

Return value

The voice number (0-23) used for KeyOn. Returns -1 if unsuccessful.

See also

[SsUtKeyOffV\(\)](#)

SsUtPitchBend

Apply a pitch bend.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtPitchBend(

short *voice*,

Voice number (0-23)

short *vabld*,

VAB number (0-31) returned by SsVabOpenHead()

short *prog*,

Program number (0-127)

short *note*,

Pitch specification in half-tone units (note number) (0-127)

short *pbend*)

Pitch-bend value (0-127)

Explanation

Applies a pitch bend (0-127, 64:center) to the voice.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtChangePitch\(\)](#), [SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtReverbOff

Turn off Reverb.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsUtReverbOff(void)

Explanation

Turns off global Reverb.

See also

[SsUtReverbOn\(\)](#)

SsUtReverbOn

Turn on Reverb.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsUtReverbOn(void)

Explanation

Turns on global Reverb at the type and depth set by SsUtSetReverbType() and SsUtSetReverbDepth().

See also

[SsUtReverbOff\(\)](#), [SsUtSetReverbType\(\)](#), [SsUtSetReverbDepth\(\)](#)

SsUtSetDetVVol

Set a detailed value of voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsUtSetDetVVol(

short *vc*, Voice number (0-23)
short *detvoll*, Detailed volume, left (0-16383)
short *detvolr*) Detailed volume, right (0-16383)

Explanation

Sets the detailed value of voice volume.

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtGetDetVVol\(\)](#), [SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsUtSetProgAtr

Set a program attribute table.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtSetProgAtr(
short *vabId*, VAB number (0-31) from the return value of the function SsVabOpen()
short *progNum*, Program number (0-127)
ProgAtr **progrptr*) Pointer to program attribute table

Explanation

Specifies a VAB number and a program number, and changes the program attribute table, *progrptr*.

- Change allowed: *mvol*, *mpan*, *prior*, *mode*, *attr*
- Change not allowed: *tones*, *reserved0*, *reserved1*, *reserved2*

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtGetProgAtr\(\)](#)

SsUtSetReverbDelay

Set a Delay volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsUtSetReverbDelay(  
short delay)           0-127
```

Explanation

Sets a delay time to be applied to Echo and Delay type reverb.

SsUtSetReverbDepth

Set a reverb depth.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

void SsUtSetReverbDepth(

short *ldepth*, Left depth (0-127)
short *rdepth*) Right depth (0-127)

Explanation

Sets a reverb depth.

See also

SsUtGetReverbType()

SsUtSetReverbFeedback

Set a feedback volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
void SsUtSetReverbFeedback(
short feedback)           Feedback (0-127)
```

Explanation

Sets a feedback volume to be applied Echo and Delay type reverb.

SsUtSetReverbType

Set reverb type.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsUtSetReverbType(
short *type*) Reverb type

Explanation

Sets reverb type.

Table 14-5: Reverb Type Overview (See Sound Delicatessen DSP)

Type	Mode	Delay time	Feedback
SPU_REV_TYPE_OFF	off	X	X
SPU_REV_TYPE_ROOM	room	X	X
SPU_REV_TYPE_STUDIO_A	studio (small)	X	X
SPU_REV_TYPE_STUDIO_B	studio (med)	X	X
SPU_REV_TYPE_STUDIO_C	studio (big)	X	X
SPU_REV_TYPE_HALL	hall	X	X
SPU_REV_TYPE_SPACE	space echo	X	X
SPU_REV_TYPE_ECHO	echo	O	O
SPU_REV_TYPE_DELAY	delay	O	O
SPU_REV_TYPE_PIPE	pipe echo	X	X

When a reverb type is set, reverb depth is automatically set to 0. Because noise occurs as soon as depth is set if data remains in the reverb work area, follow the procedure below:

```
SsUtSetReverbType (SS_REV...);  
SsUtReverbOn();
```

Wait for several seconds.

```
SsUtSetReverbDepth (64, 64);
```

Return value

The Type number that was set, if setting was correctly performed; otherwise -1.

See also

[SsUtGetReverbType\(\)](#)

SsUtSetVabHdr

Set a VAB attribute header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
short SsUtSetVabHdr(
short vabId,           VAB number (0-31) returned by SsVabOpenHead()
VabHdr *vabhdrptr)   Pointer to VAB attribute header
```

Explanation

Specifies the VAB number (the return value of SsVabOpenHead()) and changes the VAB attribute header, *vabhdrptr*.

- Setting allowed: mvol, pan, attr1, attr2 only
- Setting not allowed: form, ver, id, fsize, reserved0, ps, ts, vs, reserved 1

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtGetVabHdr\(\)](#)

SsUtSetVagAtr

Set a tone attribute table (VagAtr).

Library	Header File	Introduced	Documentation Date
libsnd.lib	libsnd.h	2.x	12/14/98

Syntax

short SsUtSetVagAtr(
 short *vabId*, VAB number (0-31) from the return value of the function SsVabOpen()
 short *progNum*, Program number (0-127)
 short *toneNum*, Tone number (0-15)
 VagAtr **vagatrptr*) Pointer to tone attribute table

Explanation

Specifies a VAB number, a program number, and a tone number, and changes a tone attribute table, *vagatrptr*.

Change allowed: Items in VagAtr that are not listed below.

Change not allowed: *prog*, *vag*, *reserved1*, *reserved2*, *reserved*[0-3]

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtGetVagAtr\(\)](#), [VagAtr\(\)](#)

SsUtSetVVol

Set voice volume.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

short SsUtSetVVol(

short *vc*, Voice number (0-23)
short *voll*, Volume, left (0-127)
short *volr*) Volume, right (0-127)

Explanation

Sets the left and right volumes of the specified voice, *vc*. Since libsnd uses an exponential volume calculation for sounds being keyed on, the input volumes *voll* and *volr* are modified as follows:

$lvol = voll * voll / 127$

$rvol = volr * volr / 127$

Return value

0 if successful, -1 if unsuccessful.

See also

[SsUtGetVVol\(\)](#), [SsUtKeyOn\(\)](#), [SsUtKeyOnV\(\)](#), [SsVoKeyOn\(\)](#)

SsVabClose

Close VAB data file.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	2.x	12/14/98

Syntax

```
void SsVabClose(  
short vab_id)           VAB data id
```

Explanation

Closes a VAB data file containing *vab_id*.

Execute it after closing the SEQ/SEP which use the specified VAB data ID.

See also

[SsVabOpen\(\)](#), [SsVabTransBody\(\)](#), [SsVabTransBodyPartly\(\)](#)

SsVabFakeBody

Assign a VAB ID to VAB data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
short SsVabFakeBody(
short vabid)          VAB id
```

Explanation

After SsVabFakeHead() has established a VAB header in RAM, this function assigns a VAB ID to VAB body data that already exists in the sound buffer, without actually performing the transfer. It isn't necessary to call SsVabTransCompleted() after calling this function.

SsVabFakeHead() and SsVabFakeBody() must be used together with SsVabOpenHeadSticky(); they cannot be used together with SsVabOpenHead().

Return value

VAB Identifying number. Returns -1 if unsuccessful.

See also

[SsVabFakeHead\(\)](#), [SsVabOpenHeadSticky\(\)](#), [SsVabTransBody\(\)](#), [SsVabTransBodyPartly\(\)](#)

SsVabFakeHead

Open a new VAB header for a given VAB body.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	2/24/99

Syntax

```
short SsVabFakeHead(
  u_char *addr,           Pointer to VH leading address
  short vabid,            Desired VAB ID. If -1, the library makes the allocation.
  u_long sbaddr)         Address of VB inside the sound buffer.
```

Explanation

Associates new VAB header data with a given VabBody.

When *vabid* is -1, the function searches for an empty VAB ID (0 - 15) and allocates it.

sbaddr is the starting address in the sound buffer of the VabBody. It is necessary to call SsVabFakeBody() next so that an actual data transfer doesn't have to be performed.

Since SsVabOpenHead(), SsVabFakeHead() and SsVabOpenHeadSticky() use and alter the system reserved area of the header list, the reserved area cannot be rewritten after header list recognition.

SsVabFakeHead() and SsVabFakeBody() must be used together with SsVabOpenHeadSticky(); they cannot be used together with SsVabOpenHead().

Return value

VAB Identifying number. Returns -1 if unsuccessful.

See also

[SsVabFakeBody\(\)](#), [SsUtGetVBaddrInSB\(\)](#), [SsVabOpenHead\(\)](#), [SsVabOpenHeadSticky\(\)](#)

SsVabOpen

Open VAB data. (Not recommended for use)

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
short SsVabOpen(
  u_char *addr,           Pointer to start address of VAB data in main storage
  VabHdr *vab_header)    Pointer to address to VAB header structure corresponding to VAB id
```

Explanation

Analyzes the VAB data header in main memory, stores the header value in *vab_header*, and returns the VAB id. It also transmits to the SPU local memory the VAG data group (wave form) data contained in VAB.

Note: This function is no longer recommended for use. Instead, use *SsVabOpenHead()* and *SsVabTransBody()* or *SsVabTransfer()*.

Return value

VAB id identifying the given VAB. On failure, -1 is returned.

See also

SsVabClose(), *SsVabOpenHead()*, *SsVabTransBody()*

SsVabOpenHead

Open a VAB header.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	2/24/99

Syntax

```
short SsVabOpenHead(
u_char *addr,      Start address of VAB header (.VH) in main memory
short vabid)       VAB ID
```

Explanation

Sets up a VAB header list in main memory so that it can be used by the Sound Library. A VAB ID may be specified to be used for opening, or if *vabid* is set to -1, the function allocates an empty VAB ID (0 - 15) if there is one. Execute *SsVabTransBody()* to transfer the VAB body to the SPU RAM, and *SsVabTransCompleted()* to confirm completion of the transfer.

This function reserves an area in SPU RAM for the VAB body in multiples of 64 bytes. Therefore, this area can be larger than the actual VAB body by up to 48 bytes (since the body is in sections of 16 bytes).

Since *SsVabOpenHead()*, *SsVabFakeHead()* and *SsVabOpenHeadSticky()* use and alter the system reserved area of the header list, the reserved area cannot be rewritten after header list recognition.

Return value

VAB identification number.

Returns -1 if unsuccessful. See error codes under *SsVabOpenHeadSticky()* for details. Also returns error if there isn't enough room in SPU RAM for the VAB.

See also

[SsVabTransBody\(\)](#), [SsVabTransBodyPartly\(\)](#), [SsVabOpenHeadSticky\(\)](#), [SsVabTransfer\(\)](#), [SsVabTransCompleted\(\)](#)

SsVabOpenHeadSticky

Open a VAB header and specify transfer address in sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	2/24/99

Syntax

```
short SsVabOpenHeadSticky(
u_char *addr,           Start address of VAB header (.VH) in main memory
short vabid,            Desired VAB ID or -1
u_long sbaddr)          Start address in sound buffer where VabBody is to be transferred
```

Explanation

Sets up a VAB header list in main memory so that it can be used by the sound library. *vabid* specifies a VAB ID to be opened; if *vabid* is set to -1, an empty VAB ID (0 - 15) is allocated, if there is one.

Set *sbaddr* to the address for transferring VabBody (.VB) to the sound buffer, within the range of 0x1010 to 0x7ffff. Take the VabBody size into consideration so that doesn't overwrite the reverb work area.

Call `SsVabTransBody()` or `SsVabTransBodyPartly()` later to transfer VabBody to *sbaddr*. Since the .VB transfer is done in 64-byte usnits, the size written to the sound buffer may be larger than the actual .VB size.

In general, you should either:

- Use `SsVabOpenHeadSticky()` and do your own memory management, or
- Use `SsVabOpenHead()` along with other routines that use libspu memory management, such as `SsVabTransfer()`, `SpuMalloc()`, `SpuFree()`, `SpuMallocWithStartAddr()`, and `SpuReserveReverbWorkArea()`.

If you combine these two approaches, the consistency of the sound buffer can't be guaranteed.

Since `SsVabOpenHead()`, `SsVabFakeHead()` and `SsVabOpenHeadSticky()` use and alter the system reserved area of the header list, the reserved area cannot be rewritten after header list recognition.

Return value

VAB identifying number.

Returns -1 if unsuccessful, in the following cases:

- The specified VabID was already open.
- The specified VabID is not within the range 0-15.
- The number of VABs allowed to be open simultaneously (16) was exceeded.
- The VAB header doesn't contain valid data.
- The size of the VabBody to be transferred will go past the end of SPU RAM.
- Could not confirm the end of the previous VabBody transfer using `SsVabTransCompleted()`.

See also

[SsVabOpenHead\(\)](#), [SsVabTransBody\(\)](#), [SsVabTransBodyPartly\(\)](#), [SsVabTransfer\(\)](#)

SsVabTransBody

Transfer sound source data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
short SsVabTransBody(  
u_char *addr,      VAB data start address  
short vabid)       VAB ID
```

Explanation

After SsVabOpenHead() recognizes a header list, this function starts transferring sound source data (VAB body) in main memory to SPU local memory. Data is transferred in 64-byte units. Use SsVabTransCompleted() to confirm transfer completion.

The starting address (*addr*) in the sound buffer into which VabBody(.VB) is transferred must be in the range 0x1010-0x7fff. It must take into account the size of the .VB, so that data is not transferred into the reverb work area.

Return value

VAB identifying number. Returns -1 if unsuccessful.

See also

[SsVabOpenHead\(\)](#), [SsVabTransBodyPartly\(\)](#), [SsVabOpenHeadSticky\(\)](#), [SsVabTransfer\(\)](#)

SsVabTransBodyPartly

Transfer sound source data in segments.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsVabTransBodyPartly(

u_char *addr,

Pointer to starting address of the segment transfer buffer

u_long bufsize,

Buffer size

short vabid)

VAB ID

Explanation

Starts transferring a VAB body in main memory, whose VAB header was opened with SsVabOpenHead(), to the sound buffer.

By continuously calling SsVabTransBodyPartly() while sequentially copying part of the sound source (VAB body) into the area indicated by *addr* with a size of *bufsize*, transfers may be made to a contiguous area within the sound buffer using only a limited area in main memory.

Since data is transferred in 64-byte units, *bufsize* must be a multiple of 64.

You must call SsVabTransCompleted() to verify whether each transfer has been completed before calling SsVabTransBodyPartly() again.

Return value

vabid, if transfer successful. Error codes are:

- 2: The size of the sound source data (VAB body) inherited from SsVabOpenHead() has not been completely transferred
- 1: Transfer failed

See also

[SsVabOpenHead\(\)](#), [SsVabTransBody\(\)](#), [SsVabOpenHeadSticky\(\)](#), [SsVabTransfer\(\)](#)

SsVabTransCompleted

Get VAB data transfer state.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

short SsVabTransCompleted(

short *immediateFlag*) Transfer status recognition flag

Explanation

Determines whether data transfer to SPU local memory has terminated.

If *immediateFlag* is SS_IMMEDIATE, the function returns the transfer state immediately. If *immediateFlag* is SS_WAIT_COMPLETED, the function loops until transfer is completed.

Return value

1 if the transfer has been completed, 0 if the transfer is ongoing.

See also

[SsVabOpenHead\(\)](#), [SsVabOpenHeadSticky\(\)](#), [SsVabTransfer\(\)](#), [SsVabTransBody\(\)](#), [SsVabTransBodyPartly\(\)](#)

SsVabTransfer

Recognize and transfer sound source data.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

```
SsVabTransfer(
  u_char *vh_addr,           Pointer to VH data
  u_char *vb_addr,           Pointer to VB data
  short vabid,               VAB ID number
  short i_flag)              = SS_IMMEDIATE...Immediately returns return value (VAB ID number)
                              = SS_WAIT_COMPLETED...Waits until transfer is completed
```

Explanation

Recognizes a sound source header list (VH data) specified by *vh_addr* and transfers sound source data (VB data) specified by *vb_addr*, to the sound buffer. The VAB ID number is specified in the argument *vabid*. When *vabid* is -1, the function searches for an empty VAB ID (0-15) and allocates it. *i_flag* determines whether the function should wait until transfer is completed or return immediately after the transfer starts, then check with `SsVabTransCompleted()`.

Return value

VAB ID number, if successful. Error codes are:

-1:	VAB ID cannot be allocated or invalid VH
-2:	Invalid VB
-3 and lower:	Other error

See also

[SsVabOpenHead\(\)](#), [SsVabOpenHeadSticky\(\)](#), [SsVabTransBody\(\)](#), [SsVabTransBodyPartly\(\)](#), [SsVabTransCompleted\(\)](#)

SsVoiceCheck

Verify tone information played by a voice that is to be modified.

Library	Header File	Introduced	Documentation Date
libsnd.lib	libsnd.h	4.1	12/14/98

Syntax

short SsVoiceCheck(
 long *voice*,
 long *vabld*,

 short *note*)

Voice number containing tone information to be verified (0-23)
The upper 8 bits of the lower 2 bytes of *vabld* contain the VAB ID returned by SsVabOpenHead(), and the lower 8 bits of the lower 2 bytes of *vabld* contain the program number containing the tone information to be verified
The note at which the tone to be verified was originally keyed on

Explanation

Verifies that the tone information played by a voice that is to be modified (key off, reverb change, pitch change etc.) is the intended tone information; that is, that the voice was not allocated to a different tone than the tone specified by *vabld* and *note*.

Should be called before each call to SsQueueRegisters() after key on of sound has occurred.

Return value

0 if successful; -1 if tone information is different than tone specified by *vabld* and *note* or *voice* is out of range.

See also

[SndRegisterAttr\(\)](#), [SndVoiceStats\(\)](#), [SndVolume2\(\)](#), [SsAllocateVoices\(\)](#), [SsBlockVoiceAllocation\(\)](#), [SsGetActualProgFromProg\(\)](#), [SsPitchFromNote\(\)](#), [SsQueueKeyOn\(\)](#), [SsQueueRegisters\(\)](#), [SsQueueReverb\(\)](#), [SsSetVoiceSettings\(\)](#), [SsUnBlockVoiceAllocation\(\)](#)

SsVoKeyOff

Key off.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

long SsVoKeyOff(

long *vab_pro*, VAB data id and program number
long *pitch*) Pitch

Explanation

Of the lower 16 bits of *vab_pro*, the upper 8 bits are used for VAB id, and the lower 8 bits specify a program number. Of the lower 16 bits of *pitch*, the upper 8 bits specify a key number in MIDI standard. To specify a finer pitch, specify a key number in the lower 8 bits of pitch in 1/128 semitones.

Return value

The keyed off voice number.

See also

[SsVoKeyOn\(\)](#)

SsVoKeyOn

Key on.

Library	Header File	Introduced	Documentation Date
<i>libsnd.lib</i>	<i>libsnd.h</i>	3.0	12/14/98

Syntax

long SsVoKeyOn(
 long *vab_pro*, VAB data id and program number
 long *pitch*, Pitch
 u_short *volL*, L channel volume
 u_short *volR*) R channel volume

Explanation

Of the lower 16 bits of *vab_pro*, the upper 8 bits are used for VAB id, and the lower 8 bits specify a program number. Of the lower 16 bits of *pitch*, the upper 8 bits specify a key number in MIDI standard. To specify a finer pitch, specify a key number in the lower 8 bits of pitch in 1/128 semitone units. The sound specified by *vab_pro* and *pitch* is keyed on at the specified *volL* and *volR*.

KeyOn with volume set to 0 is the same as SsVoKeyOff().

Return value

Voices which were keyed on. To determine if a specific voice was keyed on, AND the return value and the appropriate voice bit SPU_xxCH (xx=0-23). If the value is non-zero, the voice was keyed on.

See also

[SsVoKeyOff\(\)](#)

Chapter 15: Basic Sound Library

Table of Contents

Structures

SpuCommonAttr	15-5
SpuDecodeData	15-6
SpuEnv	15-7
SpuExtAttr	15-8
SpuLVoiceAttr	15-9
SpuReverbAttr	15-10
SpuStEnv	15-11
SpuStVoiceAttr	15-12
SpuVoiceAttr	15-13
SpuVolume	15-14

Functions

SpuClearReverbWorkArea	15-15
SpuFlush	15-16
SpuFree	15-17
SpuGetAllKeysStatus	15-18
SpuGetCommonAttr	15-19
SpuGetCommonCDMix	15-20
SpuGetCommonCDReverb	15-21
SpuGetCommonCDVolume	15-22
SpuGetCommonMasterVolume	15-23
SpuGetCommonMasterVolumeAttr	15-24
SpuGetCommonMasterVolumeX	15-25
SpuGetIRQ	15-26
SpuGetIRQAddr	15-27
SpuGetKeyStatus	15-28
SpuGetMute	15-29
SpuGetNoiseClock	15-30
SpuGetNoiseVoice	15-31
SpuGetPitchLFOVoice	15-32
SpuGetReverb	15-33
SpuGetReverbModeDelayTime	15-34
SpuGetReverbModeDepth	15-35
SpuGetReverbModeFeedback	15-36
SpuGetReverbModeParam	15-37
SpuGetReverbModeType	15-38
SpuGetReverbVoice	15-39
SpuGetTransferMode	15-40
SpuGetTransferStartAddr	15-41
SpuGetVoiceADSR	15-42
SpuGetVoiceADSRAttr	15-43
SpuGetVoiceAR	15-44
SpuGetVoiceARAttr	15-45
SpuGetVoiceAttr	15-46
SpuGetVoiceDR	15-47
SpuGetVoiceEnvelope	15-48
SpuGetVoiceEnvelopeAttr	15-49
SpuGetVoiceLoopStartAddr	15-50
SpuGetVoiceNote	15-51
SpuGetVoicePitch	15-52
SpuGetVoiceRR	15-53
SpuGetVoiceRRAttr	15-54
SpuGetVoiceSampleNote	15-55

SpuGetVoiceSL	15-56
SpuGetVoiceSR	15-57
SpuGetVoiceSRAttr	15-58
SpuGetVoiceStartAddr	15-59
SpuGetVoiceVolume	15-60
SpuGetVoiceVolumeAttr	15-61
SpuGetVoiceVolumeX	15-62
Spulnit	15-63
SpulnitHot	15-64
SpulnitMalloc	15-65
SpulsReverbWorkAreaReserved	15-66
SpulsTransferCompleted	15-67
SpuLSetVoiceAttr	15-68
SpuMalloc	15-69
SpuMallocWithStartAddr	15-70
SpuNGetVoiceAttr	15-71
SpuNSetVoiceAttr	15-72
SpuQuit	15-73
SpuRead	15-74
SpuReadDecodedData	15-75
SpuReserveReverbWorkArea	15-76
SpuRGetAllKeysStatus	15-77
SpuRSetVoiceAttr	15-78
SpuSetCommonAttr	15-79
SpuSetCommonCDMix	15-80
SpuSetCommonCDReverb	15-81
SpuSetCommonCDVolume	15-82
SpuSetCommonMasterVolume	15-83
SpuSetCommonMasterVolumeAttr	15-84
SpuSetEnv	15-85
SpuSetESA	15-86
SpuSetIRQ	15-87
SpuSetIRQAddr	15-88
SpuSetIRQCallback	15-89
SpuSetKey	15-90
SpuSetKeyOnWithAttr	15-91
SpuSetMute	15-92
SpuSetNoiseClock	15-93
SpuSetNoiseVoice	15-94
SpuSetPitchLFOVoice	15-95
SpuSetReverb	15-96
SpuSetReverbDepth	15-97
SpuSetReverbModeDelayTime	15-98
SpuSetReverbModeDepth	15-99
SpuSetReverbModeFeedback	15-100
SpuSetReverbModeParam	15-101
SpuSetReverbModeType	15-103
SpuSetReverbVoice	15-104
SpuSetTransferCallback	15-105
SpuSetTransferMode	15-106
SpuSetTransferStartAddr	15-107
SpuSetVoiceADSR	15-108
SpuSetVoiceADSRAttr	15-109
SpuSetVoiceAR	15-110
SpuSetVoiceARAttr	15-111
SpuSetVoiceAttr	15-112
SpuSetVoiceDR	15-115

SpuSetVoiceLoopStartAddr	15-116
SpuSetVoiceNote	15-117
SpuSetVoicePitch	15-118
SpuSetVoiceRR	15-119
SpuSetVoiceRRAttr	15-120
SpuSetVoiceSampleNote	15-121
SpuSetVoiceSL	15-122
SpuSetVoiceSR	15-123
SpuSetVoiceSRAttr	15-124
SpuSetVoiceStartAddr	15-125
SpuSetVoiceVolume	15-126
SpuSetVoiceVolumeAttr	15-127
SpuStart	15-128
SpuStGetStatus	15-129
SpuStGetVoiceStatus	15-130
SpuStInit	15-131
SpuStQuit	15-132
SpuStSetPreparationFinishedCallback	15-133
SpuStSetStreamFinishedCallback	15-134
SpuStSetTransferFinishedCallback	15-135
SpuStTransfer	15-136
SpuWrite	15-137
SpuWrite0	15-138
SpuWritePartly	15-139

SpuCommonAttr

Common attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    u_long mask;           Set mask
    SpuVolume mvol;        Master volume
    SpuVolume mvolmode;    Master volume mode
    SpuVolume mvolx;       Current master volume
    SpuExtAttr cd;          Cd input attributes
    SpuExtAttr ext;        External digital input attributes
} SpuCommonAttr;
```

Explanation

Used when setting/checking common attributes. The members needed for setting are set as bit values in *mask*.

See also

[SpuVolume\(\)](#), [SpuExtAttr\(\)](#), [SpuSetCommonAttr\(\)](#), [SpuGetCommonAttr\(\)](#)

SpuDecodeData

Decoded data from SPU.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	2.x	12/14/98

Structure

```
#define SPU_DECODEDATA_SIZE 0x200

typedef struct {
    short cd_left[SPU_DECODEDATA_SIZE];    CD L channel data decoded by SPU
    short cd_right[SPU_DECODEDATA_SIZE];    CD R channel data decoded by SPU
    short voice1[SPU_DECODEDATA_SIZE];      Voice 1 data decoded by SPU
    short voice3[SPU_DECODEDATA_SIZE];      Voice 3 data decoded by SPU
} SpuDecodeData;
```

Explanation

This structure describes an area used when getting CD-ROM, voice 1 and voice 3 data decoded by the SPU. Each member specifies a buffer for that type of data. Note that the sizes shown are in short integers, so total size of SpuDecodeData region is 0x1000 bytes.

When you call SpuReadDecodedData(), the SPU copies data from its buffers to the SpuDecodeData struct in main memory that you specify. It copies the data one-half buffer (0x200 bytes) at a time, and returns a code specifying which half of the buffer is currently being written to, so you can use the data from the other half.

See also

[SpuReadDecodedData\(\)](#)

SpuEnv

SPU environment attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    u_long mask;           Setting mask
    u_long queueing;       Event queueing
} SpuEnv;
```

Explanation

Used to set the basic sound library environment. Currently, only queueing of the following events can be set - Key on, Key off, Pitch LFO voice setting, Noise Voice setting, and Reverb Voice setting.

The default value state is to perform the setting of these events immediately.

See also

[SpuSetEnv\(\)](#), [SpuFlush\(\)](#)

SpuExtAttr

External input attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	12/14/98

Structure

```
typedef struct {  
    SpuVolume volume;           Volume  
    long reverb;                Reverb on/off  
    long mix;                   Mixing on/off  
} SpuExtAttr;
```

Explanation

Used when setting/checking CD and external digital input attributes.

See also

[SpuCommonAttr\(\)](#), [SpuVolume\(\)](#)

SpuLVoiceAttr

Voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	4.4	12/14/98

Structure

```
typedef struct {
    short voiceNum;           Voice number (0-23)
    short pad;
    SpuVoiceAttr attr;       Voice attributes
} SpuLVoiceAttr;
```

Explanation

Specifies voice attributes for a specific voice. An array of SpuLVoiceAttr structures is passed to SpuLSetVoiceAttr() to set attributes for multiple voices.

voiceNum specifies the voice for which the attributes should be set. *attr.voice* settings are ignored.

See also

[SpuGetVoiceAttr\(\)](#), [SpuSetVoiceAttr\(\)](#), [SpuLSetVoiceAttr\(\)](#)

SpuReverbAttr

Reverb attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Structure

```
typedef struct {  
    u_long mask;           Set mask  
    long mode;             Reverb mode  
    SpuVolume depth;       Reverb depth  
    long delay;            DelayTime (ECHO, DELAY only)  
    long feedback;         Feedback (ECHO, DELAY only)  
} SpuReverbAttr;
```

Explanation

Used when setting/checking reverb attributes. The members required at setting are set in the mask as bit values.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuSetReverbDepth\(\)](#)

SpuStEnv

SPU streaming environment attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    long size;
    long low_priority;

    SpuStVoiceAttr voice[24];
} SpuStEnv
```

Stream buffer size
Determines priority of SPU Streaming compared to other CPU processes. Default is SPU_OFF. Setting to SPU_ON lowers SPU Streaming priority.
Each stream attribute set

Explanation

Used in SPU streaming library, streaming environment and stream attribute setting.

See also

[SpuStInit\(\)](#)

SpuStVoiceAttr

SPU streaming voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    char status;           Stream status
    char pad1, pad2, pad3;  Padding
    long last_size;        Size of final data transfer ( <= (size / 2))
    u_long buf_addr;       Start address of stream buffer
    u_long data_addr;      Start address of stream data in main RAM
} SpuStVoiceAttr;
```

Explanation

Contains attributes for each stream used by the SPU streaming library. See also [SpuStEnv](#).

See also

[SpuStInit\(\)](#)

SpuVoiceAttr

Voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Structure

```
typedef struct {
    u_long voice;           Voice (low 24 bits are a bit string, 1 bit per voice )
    u_long mask;           Attributes (bit string, 1 bit per attribute)
    SpuVolume volume;      Volume
    SpuVolume volmode;     Volume mode
    SpuVolume volumex;     Current volume
    u_short pitch;         Interval (set pitch)
    u_short note;          Interval (set note)
    u_short sample_note;   Interval (set note)
    short envx;            Current envelope volume value
    u_long addr;           Waveform data start address
    u_long loop_addr;      Starting address of loop
    long a_mode;           Attack rate mode
    long s_mode;           Sustain rate mode
    long r_mode;           Release rate mode
    u_short ar;            Attack rate
    u_short dr;            Decay rate
    u_short sr;            Sustain rate
    u_short rr;            Release rate
    u_short sl;            Sustain level
    u_short adsr1;         Same value as structure VagAtr adsr1
    u_short adsr2;         Same value as structure VagAtr adsr2
} SpuVoiceAttr;
```

Explanation

Used when setting/checking the attributes of each voice. The voice number is provided/obtained from the voice bit value, and the members needed for setting are set as bit values in the mask.

Note: Constant macro names spelled SPU_ON, SPU_OFF have the same values as and are interchangeable with constant macros used in the program and spelled SpuOn, SpuOff.

See also

[SpuSetVoiceAttr\(\)](#), [SpuGetVoiceAttr\(\)](#), [SpuSetKeyOnWithAttr\(\)](#)

SpuVolume

Volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	2.x	12/14/98

Structure

```
typedef struct {
    short left;           L channel value
    short right;          R channel value
} SpuVolume;
```

Explanation

Used in attributes that require L channel/R channel values when setting/getting each voice.

See also

[SpuVoiceAttr\(\)](#), [SpuReverbAttr\(\)](#), [SpuExtAttr\(\)](#), [SpuCommonAttr\(\)](#)

SpuClearReverbWorkArea

Clear reverb work area.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuClearReverbWorkArea(
long rev_mode)           Reverb mode
```

Explanation

Clears the area occupied by the reverb work area corresponding to the reverb mode *rev_mode*.

Regardless of whether or not it is reserved at this time, the function checks to see if the area is being used.

This operation uses synchronous DMA transfer, so it blocks until finished. The time taken depends on the reverb mode; it should take a maximum of 1/5 second.

This function should not be called while another Spu transfer is occurring, as the SpuSetTransferCallback is temporarily cleared inside this function and thus the end of the pending transfer may be missed.

Return value

0 if successful. SPU_ERROR is returned if the reverb work area corresponding to *rev_mode* is in use, or if the specified reverb mode value is wrong.

See also

[SpuSetReverbModeParam\(\)](#), [SpuReserveReverbWorkArea\(\)](#), [SpuSetReverb\(\)](#), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#)

SpuFlush

Flush queued events.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
u_long SpuFlush(  
u_long ev)           Event to be flushed
```

Explanation

Flushes a queued event.

Set *ev* with bitwise inclusive ORed events to be flushed:

SPU_EVENT_KEY	Key ON/OFF
SPU_EVENT_PITCHLFO	Pitch LFO Voice Set
SPU_EVENT_NOISE	Noise Voice Set
SPU_EVENT_REVERB	Reverb Voice Set

When *ev* is set to SPU_EVENT_ALL, all events are flushed.

Return value

Bitwise inclusive ORed value of the flushed event(s).

See also

[SpuSetEnv\(\)](#), [SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#), [SpuSetPitchLFOVoice\(\)](#), [SpuSetNoiseVoice\(\)](#), [SpuSetReverbVoice\(\)](#)

SpuFree

Release area allocated in sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
void SpuFree(
u_long addr)
```

Start address of allocated area (in bytes)

Explanation

Releases area allocated in the sound buffer as indicated by the start address *addr*, and deletes that area's information from the management table.

See also

[SpuInitMalloc\(\)](#), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#)

SpuGetAllKeysStatus

Determine key on/off for all voices.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

**void SpuGetAllKeysStatus(
char *status)** Pointer to the result of checking a voice (24 bytes)

Explanation

Checks key on/key off and envelope status of all voices. *status* is a 24-byte array containing the key on/key off and envelope status of each voice.

See Table 15–1 under SpuGetKeyStatus() for information about the values that can be returned.

See also

[SpuSetKey\(\)](#), [SpuGetKeyStatus\(\)](#), [SpuRGetAllKeysStatus\(\)](#)

SpuGetCommonAttr

Check sound system attributes

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
void SpuGetCommonAttr(  
SpuCommonAttr *attr)           Pointer to attributes common to all voices
```

Explanation

Returns sound system attributes in *attr*. See `SpuSetCommonAttr()` for details.

See also

`SpuSetCommonAttr()`, `SpuCommonAttr()`

SpuGetCommonCDMix

Get CD input on/off status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuGetCommonCDMix(  
long *on_off)           CD input on/off
```

Explanation

Gets the CD input on/off status. Equivalent to getting *cd.mix* member of *SpuCommonAttr* using *SpuGetCommonAttr()*.

See also

[SpuGetCommonAttr\(\)](#), [SpuSetCommonCDMix\(\)](#)

SpuGetCommonCDReverb

Get CD input reverb on/off status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuGetCommonCDReverb(
long *on_off)           CD input reverb on/off
```

Explanation

Gets the CD input reverb on/off status. Equivalent to getting *cd.reverb* member of *SpuCommonAttr* using *SpuGetCommonAttr()*.

See also

[SpuGetCommonAttr\(\)](#), [SpuSetCommonCDReverb\(\)](#)

SpuGetCommonCDVolume

Get CD input volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuGetCommonCDVolume(
```

short *cdvolL,	CD Input volume (left)
short *cdvolR)	CD Input volume (right)

Explanation

Gets the CD input volume. Equivalent to getting *cd.volume* member of *SpuCommonAttr* using *SpuGetCommonAttr()*.

See also

SpuGetCommonAttr(), SpuSetCommonCDVolume()

SpuGetCommonMasterVolume

Get master volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

void SpuGetCommonMasterVolume(

short *mvolL, Master volume (left)

short *mvolR) Master volume (right)

Explanation

Gets master volume. Equivalent to getting *mvol* member of *SpuCommonAttr* using *SpuGetCommonAttr()*.

The value is valid only when the volume mode is 'direct mode'. Other volume modes are undefined.

When the volume mode is not 'direct mode', or to get both the volume and the volume mode at the same time, use *SpuGetCommonMasterVolumeAttr()*.

See also

[SpuGetCommonAttr\(\)](#), [SpuGetCommonMasterVolumeAttr\(\)](#), [SpuSetCommonMasterVolume\(\)](#), [SpuSetCommonMasterVolumeAttr\(\)](#)

SpuGetCommonMasterVolumeAttr

Get master volume/master volume mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

void SpuGetCommonMasterVolumeAttr(

short **mvoll*,

Master volume (left)

short **mvolR*,

Master volume (right)

short **mvolModeL*,

Master Volume Mode (left)

short **mvolModeR*)

Master Volume Mode (right)

Explanation

Gets the master volume/master volume mode. Equivalent to getting *mvol* and *mvolmode* members of the *SpuCommonAttr* using *SpuGetCommonAttr()*.

See also

[SpuGetCommonAttr\(\)](#), [SpuGetCommonMasterVolume\(\)](#), [SpuSetCommonMasterVolume\(\)](#), [SpuSetCommonMasterVolumeAttr\(\)](#)

SpuGetCommonMasterVolumeX

Get current master volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

void SpuGetCommonMasterVolumeX(

short *mvolXL, Current master volume (left)
short *mvolXR) Current master volume (right)

Explanation

Gets the current master volume. Equivalent to getting *mvolx* member of *SpuCommonAttr* using *SpuGetCommonAttr()*.

See also

[SpuGetCommonAttr\(\)](#), [SpuGetCommonMasterVolume\(\)](#), [SpuGetCommonMasterVolumeAttr\(\)](#),
[SpuSetCommonMasterVolume\(\)](#), [SpuSetCommonMasterVolumeAttr\(\)](#)

SpuGetIRQ

Check status of interrupt request.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuGetIRQ(*void*)

Explanation

Checks status of interrupt request.

Return value

SPU_ON	Interrupt request is set
SPU_OFF	Interrupt request is not set

See also

[SpuSetIRQ\(\)](#)

SpuGetIRQAddr

Check interrupt request address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

u_long SpuGetIRQAddr(*void*)

Explanation

Returns interrupt request address value.

Return value

Currently set address.

See also

[SpuSetIRQAddr\(\)](#), [SpuSetIRQ\(\)](#)

SpuGetKeyStatus

Check key on/key off status for a voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuGetKeyStatus(
u_long *voice_bit*) Checked voice

Explanation

Checks key on/key off and envelope status for a voice specified in *voice_bit* with one of the values SPU_0CH ... SPU_23CH. (If multiple bits are set, the smallest voice number set is selected.)

Return value

If successful, the current key on/key off status and envelope status of the specified voice are returned. (See the table below.) If the specified voice is incorrect, -1 is returned.

Table 15–1: Key and Envelope Status Values

Value	Status	Description
SPU_ON	Key on status Not turned off by SpuSetKey() Envelope not 0	Sound currently playing.
SPU_ON_ENV_OFF	Key on status Not turned off by SpuSetKey() Envelope 0	Sound either about to start playing, or non-looping sound expired.
SPU_OFF_ENV_ON	Key off status Turned off by SpuSetKey() Envelope not 0	Sound in release state.
SPU_OFF	Key off status Turned off by SpuSetKey() Envelope 0	Sound off.

See also

[SpuSetKey\(\)](#), [SpuGetAllKeysStatus\(\)](#)

SpuGetMute

Check sound muting status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuGetMute(void)

Explanation

Checks current sound mute on/off status.

Return value

SPU_ON	Mute off
SPU_OFF	Mute on

See also

[SpuSetMute\(\)](#)

SpuGetNoiseClock

Check noise source clock.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuGetNoiseClock(void)

Explanation

Returns the value of the noise source clock.

Return value

Current noise source clock value.

See also

[SpuSetNoiseClock\(\)](#)

SpuGetNoiseVoice

Check noise source ON/OFF for each voice

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

u_long SpuGetNoiseVoice(*void*)

Explanation

Checks current status of noise source ON/OFF for each voice.

Return value

An unsigned long with 1 bit set for each voice whose noise source is on. To check a voice, AND this value with the bit mask for the voice (SPU_0CH...SPU_23CH). If the value is non-zero, the noise source is on.

See also

[SpuSetNoiseClock\(\)](#), [SpuSetNoiseVoice\(\)](#)

SpuGetPitchLFOVoice

Check pitch LFO ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

u_long SpuGetPitchLFOVoice(*void*)

Explanation

Checks current status of pitch LFO ON/OFF for each voice.

Return value

An unsigned long with 1 bit set for each voice whose pitch LFO is on. To check a voice, AND this value with the bit mask for the voice (SPU_0CH...SPU_23CH). If the value is non-zero, the pitch LFO is on.

See also

[SpuSetPitchLFOVoice\(\)](#)

SpuGetReverb

Check reverb status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuGetReverb(void)

Explanation

Checks current reverb ON/OFF status.

Return value

SPU_ON	Reverb on
SPU_OFF	Reverb off

See also

[SpuSetReverb\(\)](#)

SpuGetReverbModeDelayTime

Get reverb mode delay time.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuGetReverbModeDelayTime(  
long *delay)           Reverb delay time
```

Explanation

Gets the reverb delay time. Equivalent to getting *delay* member of `SpuReverbAttr` using `SpuGetReverbModeParam()`.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuSetReverbModeDelayTime\(\)](#)

SpuGetReverbModeDepth

Get reverb mode depth.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

void SpuGetReverbModeDepth(

short **depthL*, Reverb depth (left)

short **depthR*) Reverb depth (right)

Explanation

Gets the reverb depth. Equivalent to getting *depth* member of *SpuReverbAttr* using *SpuGetReverbModeParam()*.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuSetReverbModeDepth\(\)](#)

SpuGetReverbModeFeedback

Get reverb mode feedback.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuGetReverbModeFeedback(
long *feedback)           Reverb feedback
```

Explanation

Gets the reverb feedback. Equivalent to getting *feedback* member of *SpuReverbAttr* using *SpuGetReverbModeParam()*.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuSetReverbModeFeedback\(\)](#)

SpuGetReverbModeParam

Check reverb mode and parameters.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
void SpuGetReverbModeParam(  
SpuReverbAttr *attr)           Pointer to reverb attributes
```

Explanation

Gets currently set reverb mode and parameters. For details see `SpuSetReverbModeParam()`.

See also

`SpuSetReverbModeParam()`, `SpuReverbAttr()`

SpuGetReverbModeType

Get reverb mode type.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuGetReverbModeType(  
long *type)                Reverb mode type
```

Explanation

Gets the reverb mode type. Equivalent to getting *mode* member of *SpuReverbAttr* using *SpuGetReverbModeParam()*.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuSetReverbModeType\(\)](#)

SpuGetReverbVoice

Check reverb ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

u_long SpuGetReverbVoice(*void*)

Explanation

Checks current reverb ON/OFF status for each voice.

Return value

An unsigned long with 1 bit set for each voice whose reverb status is on. To check a voice, AND this value with the bit mask for the voice (SPU_0CH...SPU_23CH). If the value is non-zero, reverb is on.

See also

[SpuSetReverbVoice\(\)](#)

SpuGetTransferMode

Get sound buffer transfer mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuGetTransferMode(void)

Explanation

Returns current value of the transfer mode when transferring from main memory to the sound buffer.

Return value

Current setting of transfer mode:

SPU_TRANSFER_BY_DMA	DMA transfer setting
SPU_TRANSFER_BY_IO	I/O transfer setting

See also

[SpuSetTransferMode\(\)](#), [SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#)

SpuGetTransferStartAddr

Get sound buffer transfer start address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

u_long SpuGetTransferStartAddr(*void*)

Explanation

Returns current start address for transferring between main memory and the sound buffer.

Return value

Current sound buffer transfer start address.

See also

[SpuSetTransferStartAddr\(\)](#), [SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#), [SpuRead\(\)](#), [SpuReadDecodedData\(\)](#)

SpuGetVoiceADSR

Get ADSR.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceADSR(
int voiceNum,           Voice number (0 - 23)
u_short *AR,            ADSR attack rate
u_short *DR,            ADSR decay rate
u_short *SR,            ADSR sustain rate
u_short *RR,            ADSR release rate
u_short *SL)            ADSR sustain level
```

Explanation

Gets each ADSR attribute used in the voice. Equivalent to getting SpuVoiceAttr members *ar*, *dr*, *sr*, *rr*, and *sl* using SpuGetVoiceAttr().

The values are valid only for the following attack, sustain, and release rate modes. For other modes, the values are undefined.

- Attack Rate Mode: SPU_VOICE_LINEARIncN (Linear Increase)
- Sustain Rate Mode: SPU_VOICE_LINEARDecN (Linear Decrease)
- Release_Rate_Mode: SPU_VOICE_LINEARDecN_(Linear Decrease)

To get multiple rates at the same time, use SpuSetVoiceADSRAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceAR\(\)](#), [SpuGetVoiceDR\(\)](#), [SpuGetVoiceSR\(\)](#), [SpuGetVoiceRR\(\)](#), [SpuGetVoiceSL\(\)](#)

SpuGetVoiceADSRAttr

Get ADSR rates and modes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceADSRAttr(
  int voiceNum,           Voice number (0 - 23)
  u_short *AR,            ADSR attack rate
  u_short *DR,            ADSR decay rate
  u_short *SR,            ADSR sustain rate
  u_short *RR,            ADSR release rate
  u_short *SL,            ADSR sustain level
  long *ARmode,           ADSR attack rate mode
  long *SRmode,           ADSR sustain rate mode
  long *RRmode)           ADSR release rate mode
```

Explanaton

Gets each ADSR attribute used in the voice. Equivalent to getting SpuVoiceAttr members *ar*, *dr*, *sr*, *rr*, *sl*, *a_mode*, *s_mode*, and *r_mode* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceADSR\(\)](#), [SpuGetVoiceAR\(\)](#), [SpuGetVoiceDR\(\)](#), [SpuGetVoiceSR\(\)](#), [SpuGetVoiceRR\(\)](#), [SpuGetVoiceSL\(\)](#), [SpuGetVoiceARAttr\(\)](#), [SpuGetVoiceSRAttr\(\)](#), [SpuGetVoiceRRAttr\(\)](#)

SpuGetVoiceAR

Get ADSR attack rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceAR(

int *voiceNum*,

Voice number (0 - 23)

u_short **AR*)

ADSR attack rate

Explanation

Gets ADSR attack rate for a voice. Equivalent to getting `SpuVoiceAttr` member *ar* using `SpuGetVoiceAttr()`.

The value is valid only when the attack rate mode is `SPU_VOICE_LINEARIncN` (Linear Increase). For other modes, the value is undefined.

To get both attack rate and attack rate mode at the same time, use `SpuGetVoiceARAttr()`.

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceARAttr\(\)](#)

SpuGetVoiceARAttr

Get ADSR attack rate / attack rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceARAttr(  
int voiceNum,           Voice number (0 - 23)  
u_short *AR,           ADSR attack rate  
long *ARmode)          ADSR attack rate mode
```

Explanation

Gets ADSR attack rate and attack rate mode for a voice. Equivalent to getting SpuVoiceAttr members *ar* and *a_mode* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceAR\(\)](#)

SpuGetVoiceAttr

Get voice attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

**void SpuGetVoiceAttr(
[SpuVoiceAttr](#) *attr)** Pointer to voice attributes

Explanation

Gets the attributes for a single voice that you specify in *attr.voice* (with one of the values SPU_0CH ... SPU_23CH). All the SpuVoiceAttr structure members are returned in *attr* except *attr.mask*. See SpuSetVoiceAttr() for the details of these attributes.

See also

[SpuSetVoiceAttr\(\)](#), [SpuRSetVoiceAttr\(\)](#), [SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#), [SpuVoiceAttr\(\)](#)

SpuGetVoiceDR

Get ADSR decay rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceDR(  
int voiceNum,           Voice number (0 - 23)  
u_short *DR)            ADSR decay rate
```

Explanation

Gets ADSR decay rate for voice. Equivalent to getting SpuVoiceAttr member *dr* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#)

SpuGetVoiceEnvelope

Get current envelope value.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceEnvelope(

int *voiceNum*,

Voice number (0 - 23)

short **envx*)

Current envelope value

Explanation

Gets the current envelope value for a voice. Equivalent to getting the `SpuVoiceAttr` member *envx*, using `SpuGetVoiceAttr()`.

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#)

SpuGetVoiceEnvelopeAttr

Get current envelope value and key ON/OFF status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceEnvelopeAttr(

int *voiceNum*,

long **keyStat*,

short **envx*)

Voice number (0 - 23)

Status of voice envelope and key ON/OFF

Current envelope value

Explanation

Gets the current envelope value, key ON/OFF and envelope status for a voice.

Refer to Table 15–1 in under `SpuGetKeyStatus()` for the key ON/OFF and envelope status values that can be specified in *keyStat*.

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceEnvelope\(\)](#), [SpuSetKey\(\)](#), [SpuGetAllKeysStatus\(\)](#), [SpuRGetAllKeysStatus\(\)](#)

SpuGetVoiceLoopStartAddr

Get loop start address of waveform data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceLoopStartAddr(
  int voiceNum,           Voice number (0 - 23)
  u_long *loopStartAddr)  Loop start address
```

Explanation

Gets loop start address of waveform data in the sound buffer. Equivalent to getting SpuVoiceAttr member *loop_addr* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetTransferStartAddr\(\)](#)

SpuGetVoiceNote

Get interval (note specification).

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

void SpuGetVoiceNote(
int *voiceNum*, Voice number (0 - 23)
u_short **note*) Interval (note specification)

Explanation

Gets Voice Interval (Note Specification). Equivalent to getting SpuVoiceAttr member *note* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceSampleNote\(\)](#), [SpuGetVoiceSampleNote\(\)](#)

SpuGetVoicePitch

Get interval (pitch specification).

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

void SpuGetVoicePitch(
int *voiceNum*, Voice number (0 - 23)
u_short **pitch*) Interval (pitch specification)

Explanation

Gets voice interval (pitch specification). Equivalent to getting SpuVoiceAttr member *pitch* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#)

SpuGetVoiceRR

Get ADSR release rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceRR(
int voiceNum,           Voice number (0 - 23)
u_short *RR)            ADSR release rate
```

Explanation

Gets ADSR release rate for a voice. Equivalent to getting SpuVoiceAttr member *rr* using SpuGetVoiceAttr().

The value is valid only when the release rate mode is SPU_VOICE_LINEARDecN (Linear Decrease mode). For other release rate modes, the value is undefined.

To get release rate and release rate mode at the same time, use SpuGetVoiceRRAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceRRAttr\(\)](#)

SpuGetVoiceRRAttr

Get ADSR release rate / release rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceRRAttr(

int *voiceNum*,

Voice number (0 - 23)

u_short **RR*,

ADSR release rate

long **RRmode*)

ADSR release rate mode

Explanation

Gets ADSR release rate / ADSR release rate mode for a voice. Equivalent to getting `SpuVoiceAttr` members *rr* and *r_mode* using `SpuGetVoiceAttr()`.

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceRR\(\)](#)

SpuGetVoiceSampleNote

Get waveform data sample note.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

void SpuGetVoiceSampleNote(
int *voiceNum*, Voice number (0 - 23)
u_short **sampleNote*) Sets waveform data sample note

Explanation

Gets waveform data sample note. Equivalent to getting SpuVoiceAttr member *sample_note* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceNote\(\)](#)

SpuGetVoiceSL

Get ADSR sustain level.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceSL(

int *voiceNum*,

Voice number (0 - 23)

u_short **SL*)

ADSR sustain level

Explanation

Gets ADSR sustain level. Equivalent to getting SpuVoiceAttr member *s/* using SpuGetVoiceAttr().

See also

SpuGetVoiceAttr(), SpuNGetVoiceAttr(), SpuGetVoiceRRAttr()

SpuGetVoiceSR

Get ADSR sustain rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceSR(
int voiceNum,           Voice number (0 - 23)
u_short *SR)           ADSR sustain rate
```

Explanation

Gets ADSR sustain rate in a voice. Equivalent to getting SpuVoiceAttr member *sr* using SpuGetVoiceAttr().

The value is valid only when sustain rate mode is SPU_VOICE_LINEARDecN (Linear Decrease mode). For other sustain rate modes, the value is undefined.

To get both sustain rate and sustain rate mode at the same time, use SpuGetVoiceSRAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceSRAttr\(\)](#)

SpuGetVoiceSRAttr

Get ADSR sustain rate and sustain rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceSRAttr(

int *voiceNum*,

Voice number (0 - 23)

u_short **SR*,

ADSR sustain rate

long **SRmode*)

ADSR sustain rate mode

Explanation

Gets ADSR sustain rate and ADSR sustain rate mode for voice. Equivalent to getting `SpuVoiceAttr` members `sr` and `s_mode` using `SpuGetVoiceAttr()`.

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceSR\(\)](#)

SpuGetVoiceStartAddr

Get address of waveform data in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuGetVoiceStartAddr(
int voiceNum,           Voice number (0 - 23)
u_long *startAddr)      Waveform data start address
```

Explanation

Gets start address of waveform data in the sound buffer. Equivalent to getting SpuVoiceAttr member *addr* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetTransferStartAddr\(\)](#)

SpuGetVoiceVolume

Get voice volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceVolume(

int *voiceNum*,

Voice Number (0 - 23)

short **volumeL*,

Volume (Left)

short **volumeR*)

Volume (Right)

Explanation

Gets voice volume. Equivalent to getting `SpuVoiceAttr` member *volume* using `SpuGetVoiceAttr()`.

The value is valid only when the volume mode is "direct mode"; otherwise, it is undefined.

When the volume mode is not "direct mode", or to get both volume and volume mode at the same time, use `SpuGetVoiceVolumeAttr()`.

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceVolumeAttr\(\)](#)

SpuGetVoiceVolumeAttr

Get volume/volume mode.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

```
void SpuGetVoiceVolumeAttr(  
int voiceNum,           Voice Number (0 - 23)  
short *volumeL,         Volume (Left)  
short *volumeR,         Volume (Right)  
short *volModeL,        Volume mode (Left)  
short *volModeR)        Volume mode (Right)
```

Explanation

Gets voice volume and volume mode for a voice. Equivalent to getting SpuVoiceAttr members *volume* and *volmode* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceVolume\(\)](#)

SpuGetVoiceVolumeX

Get current volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuGetVoiceVolumeX(

int *voiceNum*, Voice Number (0 - 23)

short **volumeXL*, Current volume (Left)

short **volumeXR*) Current volume (Right)

Explanation

Gets current volume for a voice. Equivalent to getting SpuVoiceAttr member *volumex* using SpuGetVoiceAttr().

See also

[SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#), [SpuGetVoiceVolume\(\)](#), [SpuGetVoiceVolumeAttr\(\)](#)

Spulnit

Initialize SPU.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	12/14/98

Syntax

void Spulnit(void)

Explanation

Initializes SPU. Called only once within the program. After initialization, the state of the SPU is:

- Master volume is 0 for both L/R
- Reverb is off; reverb work area is not reserved
- Reverb depth and volume are 0 for both L/R
- Sound buffer transfer mode is DMA transfer
- For all voices: Key off
- For all voices: Pitch LFO, noise, reverb functions not set
- CD input volume is 0 for both L/R
- External digital input volume is 0 for both L/R
- DMA transfer initialization set

The status of the sound buffer is indeterminate after initialization.

See also

[SpulnitHot\(\)](#), [SpuStart\(\)](#), [SpuQuit\(\)](#)

SpulnitHot

Initialize SPU (hot reset); preserves sound buffer status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	12/14/98

Syntax

void SpulnitHot(void)

Explanation

Initializes SPU. Call SpulnitHot() when you initialize the sound system and want to preserve the sound buffer status in a child process.

After initialization, status is the same as Spulnit() except:

- Voice sample notes not cleared
- CD volume not cleared
- Pitch LFO/ noise voice not cleared
- Voice info (volume, pitch, start address, ADSR) not cleared

Voices are keyed off, however.

See also

[Spulnit\(\)](#), [SpuStart\(\)](#), [SpuQuit\(\)](#)

SpuInitMalloc

Initialize sound buffer memory management mechanism.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuInitMalloc(
  long num,           Maximum number of times memory is allocated
  char *top)          Pointer to the start address of the memory management table
```

Explanation

Initializes memory management for the sound buffer. You specify *n* as the maximum number of memory blocks that will be allocated, and an area pointed to by *top* to hold a memory management table, which stores information about each block.

The size of the area pointed to by *top* must be:

$(\text{SPU_MALLOC_RECSIZ} \cdot (\text{num} + 1))$ bytes

For example, to allow for 10 `SpuMalloc()` calls:

```
char rec[SPU_MALLOC_RECSIZ * (10 + 1)];
SpuInitMalloc (10,          /*10 SpuMalloc calls can be made*/
  rec);                    /*memory management block*/
```

Return value

The number of memory management blocks specified.

See also

`malloc()` (See `libmath`), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#), [SpuFree\(\)](#)

SpulsReverbWorkAreaReserved

Check if reverb work area is / can be reserved.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpulsReverbWorkAreaReserved(
long on_off)
```

Contents of the checking process

Explanation

Checks to see if the reverb work area corresponding to the current reverb mode is reserved or can be reserved, depending on the value of *on_off*:

- SPU_DIAG: Using sound buffer memory management mechanism information, SPU_DIAG checks to see whether or not the reverb work area is an area allocated by `SpuMalloc()`/`SpuMallocWithStartAddr()`.
- SPU_CHECK: Checks current reverb work area reserve status.

Return value

SPU_ON if the reverb work area is reserved (when *on_off* is SPU_CHECK) or can be reserved (when *on_off* is SPU_DIAG).

SPU_OFF if the reverb work area isn't reserved (when *on_off* is SPU_CHECK) or can't be reserved (when *on_off* is SPU_DIAG).

See also

[SpuReserveReverbWorkArea\(\)](#), [SpuSetReverbModeParam\(\)](#), [SpuSetReverb\(\)](#), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#)

SpulsTransferCompleted

Check completion of transfer to the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpulsTransferCompleted(
long flag)           Check flag
```

Explanation

Checks whether transfer is completed or waits for completion, depending on value of *flag*:

SPU_TRANSFER_WAIT	Wait until transfer ends
SPU_TRANSFER_PEEK	Check whether transfer has ended and return result
SPU_TRANSFER_GLANCE	Same as SPU_TRANSFER_PEEK

This function doesn't work (and returns 1) when a callback function is set with `SpuSetTransferCallback()` and started at the completion of DMA transfer.

Return value

1 : transfer completed; 0 : transfer not completed.

If *flag* = SPU_TRANSFER_WAIT, waits until transfer ends and always returns 1. If transfer mode is SPU_TRANSFER_BY_IO, 1 is returned immediately.

See also

[SpuWrite\(\)](#), [SpuWritePartly\(\)](#), [SpuRead\(\)](#), [SpuReadDecodedData\(\)](#), [SpuSetTransferCallback\(\)](#)

SpuLSetVoiceAttr

Sets attributes for multiple voices.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	4.4	12/14/98

Syntax

```
void SpuLSetVoiceAttr(  
int num                Number of elements in argList array  
SpuVoiceAttr *argList) Address of voice attribute array
```

Explanation

Collectively sets the voice attributes for each individual voice specified in *argList*.

Although this function is equivalent to executing *SpuNSetVoiceAttr()* for each voice, processing is faster with *SpuLSetVoiceAttr()*.

The *argList[x].attr.voice* specification is ignored and *argList[x].attr* is set for the voices specified in *argList[x].voiceNum*.

Set *argList[x].attr.mask* with bitwise inclusive ORed attributes to specify attributes to be set. When *argList[x].attr.mask* is 0, all attributes are set. See Table 15–6 under *SpuSetVoiceAttr()* for information about the attributes that can be set.

SpuMalloc

Allocate an area in the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

**long SpuMalloc(
long *size*)** Size of area allocated (in bytes)

Explanation

Allocate an area of *size* bytes in the sound buffer.

Failure occurs if:

- The requested size cannot be continuously allocated.
- The only area that satisfies the requested size is part or all of a reverb work area already allocated by `SpuReserveReverbWorkArea()`.

Return value

The starting address of the allocated area, if successful; -1 if unsuccessful.

See also

[SpuInitMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#), [SpuFree\(\)](#), [SpuSetTransferStartAddr\(\)](#), [SpuWrite\(\)](#), [SpuReserveReverbWorkArea\(\)](#), [SpuSetReverb\(\)](#), [SpuSetReverbModeParam\(\)](#)

SpuMallocWithStartAddr

Allocate an area from an address in sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuMallocWithStartAddr(

u_long *addr*,

Allocated area starting address (in bytes)

long *size*)

Size of allocated area (in bytes)

Explanation

Allocates an area in the sound buffer of *size* bytes starting from the address *addr*. (The allocatable area is 0x01010 - 0x7fff.) If *addr* is in an area already allocated, an area of *size* bytes is allocated starting from the nearest empty area after *addr*.

Failure occurs if:

- The requested size cannot be continuously allocated.
- The only area that satisfies the requested size is part or all of a reverb work area already allocated by `SpuReserveReverbWorkArea()`.

Return value

The starting address of the allocated area, if successful; -1 if unsuccessful.

See also

[SpuInitMalloc\(\)](#), [SpuMalloc\(\)](#), [SpuFree\(\)](#), [SpuSetTransferStartAddr\(\)](#), [SpuWrite\(\)](#), [SpuReserveReverbWorkArea\(\)](#), [SpuSetReverb\(\)](#), [SpuSetReverbModeParam\(\)](#)

SpuNGetVoiceAttr

Get voice attributes.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

```
void SpuNGetVoiceAttr(  
int voiceNum,           Voice number (0 - 23)  
SpuVoiceAttr *attr)    Voice attribute
```

Explanation

Gets attributes for voice *voiceNum*. All attributes of the *attr* structure are returned except *attr.mask*. Refer to Table 15–6 under SpuSetVoiceAttr() for details of each attribute.

See also

SpuGetVoiceAttr(), SpuSetVoiceAttr(), SpuNSetVoiceAttr(), SpuRSetVoiceAttr(), SpuSetKey(), SpuSetKeyOnWithAttr(), SpuGetVoiceVolume(), SpuGetVoiceVolumeAttr(), SpuGetVoiceVolumeX(), SpuGetVoicePitch(), SpuGetVoiceNote(), SpuGetVoiceSampleNote(), SpuGetVoiceEnvelope(), SpuGetVoiceStartAddr(), SpuGetVoiceLoopStartAddr(), SpuGetVoiceAR(), SpuGetVoiceDR(), SpuGetVoiceSR(), SpuGetVoiceRR(), SpuGetVoiceSL(), SpuGetVoiceARAttr(), SpuGetVoiceSRAttr(), SpuGetVoiceRRAttr(), SpuGetVoiceADSR(), SpuGetVoiceADSRAttr(), SpuGetVoiceEnvelopeAttr()

SpuNSetVoiceAttr

Set attributes for a voice.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

```
void SpuNSetVoiceAttr(  
int voiceNum,           Voice number (0 - 23)  
SpuVoiceAttr *attr)    Voice attribute
```

Explanation

Sets the attributes for a specific voice, specified by *voiceNum*.
Specify which attributes are to be set by setting the appropriate bits in *attr.mask*. (If *attr.mask* is 0, all attributes are set.) See Table 15–6 under *SpuSetVoiceAttr()* for information about specific attributes.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#),[SpuRSetVoiceAttr\(\)](#), [SpuGetVoiceAttr\(\)](#), [SpuNGetVoiceAttr\(\)](#),
[SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#), [SpuSetVoiceVolume\(\)](#), [SpuSetVoiceVolumeAttr\(\)](#), [SpuSetVoicePitch\(\)](#),
[SpuSetVoiceNote\(\)](#), [SpuSetVoiceSampleNote\(\)](#), [SpuSetVoiceStartAddr\(\)](#), [SpuSetVoiceLoopStartAddr\(\)](#),
[SpuSetVoiceAR\(\)](#), [SpuSetVoiceDR\(\)](#), [SpuSetVoiceSR\(\)](#),[SpuSetVoiceRR\(\)](#), [SpuSetVoiceSL\(\)](#),
[SpuSetVoiceARAttr\(\)](#), [SpuSetVoiceSRAttr\(\)](#), [SpuSetVoiceRRAttr\(\)](#),[SpuSetVoiceADSR\(\)](#),
[SpuSetVoiceADSRAttr\(\)](#)

SpuQuit

Terminate SPU processing.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

void SpuQuit(void)

Explanation

Terminates SPU processing, and releases events allocated by SPUNit(), so that the maximum number of events is not exceeded. This is particularly important when other processes may be calling SpuInit(), such as child processes, or in a game that may be part of a demo disk

See also

[SpuInit\(\)](#), [SpuInitHot\(\)](#)

SpuRead

Transfer data from sound buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
u_long SpuRead(  
  u_char *addr,           Pointer to transfer data start address in main memory  
  u_long size)             Transferred data size (in bytes)
```

Explanation

Transfers *size* bytes of data from the sound buffer to main memory address *addr*. *addr* must be a global variable or a variable in a heap area that was allocated by a function such as `malloc()`. It can't be the address of a variable declared on the stack in a function.

To confirm transfer completion, either call `SpulsTransferCompleted()` or set the DMA transfer completion callback function in advance using `SpuSetTransferCallback()`.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. Therefore, if the arguments aren't multiples of 64, it's possible to damage the data in the SPU memory.

Return value

Size of data transferred. (If *size* is larger than 512 KB, the actual transferred size is returned.)

See also

[SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#), [SpuSetTransferStartAddr\(\)](#), [SpuGetTransferStartAddr\(\)](#), [SpulsTransferCompleted\(\)](#), [SpuSetTransferCallback\(\)](#)

SpuReadDecodedData

Transfer sound data decoded by SPU from sound buffer to main memory.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuReadDecodedData(
 [SpuDecodedData](#) *d_data, Start address of SpuDecodeData structure in main memory
 long flag) SPU_CDONLY: Set transfer of only CD input
 SPU_VOICEONLY: Set transfer of only Voice 1, 3
 SPU_ALL: Set transfer of all data

Explanation

Transfers waveform data decoded by the SPU from the sound buffer to main memory.

The SPU automatically writes CD input data and Voice 1 and Voice 3 envelope data to the 0x1000-byte area at the beginning of the sound buffer, 16 bits at a time at each SPU tick (44.1kHz). Each type of sound data has a 0x400 byte buffer, divided into two halves of 0x200 bytes each. The SPU writes one half at a time.

Table 15–2: Arrangement of Data

Map (bytes)	Data Contents
0x000 - 0x3ff	CD Left channel
0x200 - 0x7ff	CD Right channel
0x400 - 0xbff	Voice 1
0x600 - 0xfff	Voice 3

The main memory address storing the transfer data must be either an address of a global variable or an address of an allocated variable in the heap allocated by a function such as malloc(). It may not address a stack area (address of an auto variable) declared in a function.

In order to confirm transfer completion, set the DMA transfer completion callback function in advance using SpuSetTransferCallback().

Return value

The buffer area currently being written to; the data that can actually be used is in the other area.

SPU_DECODE_FIRSTHALF Writes the first half of data
SPU_DECODE_SECONDDHALF Writes the second half of data

See also

[SpuRead\(\)](#), [SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#), [SpuSetTransferStartAddr\(\)](#),
[SpuGetTransferStartAddr\(\)](#), [SpulsTransferCompleted\(\)](#), [SpuSetTransferCallback\(\)](#)

SpuReserveReverbWorkArea

Reserve/release reverb work area.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuReserveReverbWorkArea(
long on_off)           Reserve/release flag
```

Explanation

Reserves the current reverb work area corresponding to the current reverb mode, so that it can't be allocated by `SpuMalloc()`/`SpuMallocWithStartAddr()`, or releases it so that it can be allocated.

on_off specifies which action is performed:

- **SPU_ON**
Reserves the reverb work area so that it can't be allocated by `SpuMalloc()`/`SpuMallocWithStartAddr()` (without regard to reverb ON/OFF).
If the area has already been allocated by `SpuMalloc()` / `SpuMallocWithStartAddr()`, it is not reserved for reverb and **SPU_OFF** is returned.
- **SPU_OFF**
Releases the reverb work area so that it can be allocated by `SpuMalloc()` / `SpuMallocWithStartAddr()`.
Releases it regardless of reverb ON/OFF; reverb must have been turned off beforehand.

Return value

The value of *on_off*, except: if *on_off* is **SPU_ON** and the reverb work area has already been allocated by `SpuMalloc()`/`SpuMallocWithStartAddr()`, **SPU_OFF** is returned.

See also

[SpulsReverbWorkAreaReserved\(\)](#), [SpuSetReverbModeParam\(\)](#), [SpuSetReverb\(\)](#), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#)

SpuRGetAllKeysStatus

Check key on/key off for a range of voices.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.1	12/14/98

Syntax

long SpuRGetAllKeysStatus(
 long *min*, Lower limit of the voice number to be checked
 long *max*, Upper limit of the voice number to be checked
 char **status*) Pointer to the result of checking a voice

Explanation

Checks key on/key off and envelope status of all voices in the range *min* to *max*.

status points to an array of 24 bytes, each containing the status value for a voice. See Table 15–1 under SpuGetKeyStatus() for a description of possible status values.

Return value

SPU_INVALID_ARGS Invalid voice range
SPU_SUCCESS Keys status contained in *status*[24].

See also

[SpuSetKey\(\)](#), [SpuGetKeyStatus\(\)](#)

SpuRSetVoiceAttr

Set attributes of a range of voices.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.1	12/14/98

Syntax

long SpuRSetVoiceAttr(
long *min*,
long *max*,
[SpuVoiceAttr](#) **attr*)

Lower limit of the voice number to be checked
Upper limit of the voice number to be checked
Pointer to voice attributes

Explanation

Sets attributes for each voice in the range specified by *min* and *max*. You can specify voices within the range by setting the bit values SPU_0CH...SPU_23CH in *attr.voice*.

Specify which attributes are to be set by setting the appropriate bits in *attr.mask*. (If *attr.mask* is 0, all attributes are set.) See Table 15–6 under SpuSetVoiceAttr() for information about specific attributes.

Return value

SPU_INVALID_ARGS
SPU_SUCCESS

Invalid voice range.
Voice attributes set for specified range.

See also

[SpuGetVoiceAttr\(\)](#), [SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#)

SpuSetCommonAttr

Set sound system attributes

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	8/9/99

Syntax

void SpuSetCommonAttr(
[SpuCommonAttr](#) *attr) Pointer to attributes common to all voices

Explanation

Sets sound system attributes. Specify the attributes (members of *attr*) by ORing together the terms shown below in *attr.mask*. If *attr.mask* is 0, all attributes are set.

Table 15–3

Attribute	Description
SPU_COMMON_MVOLL	Master volume (left)
SPU_COMMON_MVOLR	Master volume (right)
SPU_COMMON_MVOLMODEL	Master volume mode (left)
SPU_COMMON_MVOLMODER	Master volume mode (right)
SPU_COMMON_CDVOLL	CD input volume (left)
SPU_COMMON_CDVOLR	CD input volume (right)
SPU_COMMON_CDREV	CD input reverb ON/OFF
SPU_COMMON_CDMIX	CD input ON/OFF
SPU_COMMON_EXTVOLL	External digital input volume (left)
SPU_COMMON_EXTVOLR	External digital input volume (right)
SPU_COMMON_EXTREV	External digital input reverb ON/OFF
SPU_COMMON_EXTMIX	External digital input ON/OFF

The individual parameters that can be set are:

- Master Volume (*attr.mvol*) and Master Volume Mode (*attr.mvolmode*)
 Left and right are set independently. The volume range obtainable and the various modes are the same as the settings for each voice; see Table 15–7 under *SpuSetVoiceAttr()*.
- CD Input Volume (*attr.cd.volume*)
 Set independently for left and right in the range -0x8000 - 0x7fff. If volume is negative, the phase is inverted.
- CD Input Reverb On/Off (*attr.cd.reverb*)
 SPU_ON = reverb on; SPU_OFF = reverb off
- CD Input Mixing On/Off (*attr.cd.mix*)
 SPU_ON = mixing on; SPU_OFF = mixing off. CD input is not output unless mixing is on.
- External Digital Input Volume (*attr.ext.volume*)
 Set independently for left and right in the range -0x8000 - 0x7fff. If volume is negative, the phase is inverted.
- External Digital Input Reverb On/Off (*attr.ext.reverb*)
 SPU_ON = reverb on; SPU_OFF = reverb off.
- External Digital Input Mixing On/Off (*attr.ext.mix*)
 SPU_ON = mixing on; SPU_OFF = mixing off. External digital input is not output unless mixing is on.

See also

[SpuGetCommonAttr\(\)](#), [SpuSetVoiceAttr\(\)](#)

SpuSetCommonCDMix

Set CD input ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuSetCommonCDMix(  
long on_off)           CD Input on/off
```

Explanation

Turns CD input mixing on/off. Equivalent to the SPU_COMMON_CDMIX SpuSetCommonAttr() mask setting. See SpuSetCommandAttr() for values for *on_off*.

See also

[SpuSetCommonAttr\(\)](#), [SpuGetCommonCDMix\(\)](#)

SpuSetCommonCDReverb

Set CD input reverb ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuSetCommonCDReverb(
long on_off)           CD Input reverb on/off
```

Explanation

Turns CD input reverb on/off. Equivalent to the SPU_COMMON_CDREV mask setting of `SpuSetCommonAttr()`. See `SpuSetCommandAttr()` for values for *on_off*.

See also

[SpuSetCommonAttr\(\)](#), [SpuGetCommonCDReverb\(\)](#)

SpuSetCommonCDVolume

Set CD input volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

void SpuSetCommonCDVolume(

short *cdvolL*,

CD input volume (left)

short *cdvolR*)

CD input volume (right)

Explanation

Sets the CD input volume. Equivalent to the SPU_COMMON_CDVOLL and SPU_COMMON_CDVOLR mask settings of SpuSetCommonAttr(). See SpuSetCommandAttr() for values for *cdvolL* and *cdvolR*.

See also

[SpuSetCommonAttr\(\)](#), [SpuGetCommonCDVolume\(\)](#)

SpuSetCommonMasterVolume

Set master volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuSetCommonMasterVolume(
short mvoll,           Master volume (left)
short mvolr)          Master volume (right)
```

Explanation

Sets the master volume. Equivalent to the SPU_COMMON_MVOLL and SPU_COMMON_MVOLR mask settings of SpuSetCommonAttr().

The volume mode is 'direct mode' and the range of the values which can be set to the *mvoll* and *mvolr* volumes is equal to that of the 'direct mode' in SpuSetVoiceAttr() (-0x4000 to 0x3fff).

To set both volume and volume mode simultaneously, use SpuSetCommonMasterVolumeAttr().

See SpuSetVoiceAttr() for values for *mvoll* and *mvolr*.

See also

[SpuSetCommonAttr\(\)](#), [SpuSetVoiceAttr\(\)](#), [SpuSetCommonMasterVolumeAttr\(\)](#),
[SpuGetCommonMasterVolume\(\)](#), [SpuGetCommonMasterVolumeAttr\(\)](#)

SpuSetCommonMasterVolumeAttr

Set master volume/master volume mode.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.7	12/14/98

Syntax

```
void SpuSetCommonMasterVolumeAttr(  
short mvolL,           Master volume (left)  
short mvolR,           Master volume (left)  
short mvolModeL,       Master volume mode (left)  
short mvolModeR)       Master volume mode (right)
```

Explanation

Sets the master volume and master volume mode. Equivalent to the SPU_COMMON_MVOLL / SPU_COMMON_MVOLR / SPU_COMMON_MVOLMODEL / SPU_COMMON_MVOLMODER mask settings of SpuSetCommonAttr().

See SpuSetVoiceAttr() for values for mvolModeL, mvolModeR, mvolL, and mvolR.

See also

[SpuSetCommonAttr\(\)](#), [SpuSetVoiceAttr\(\)](#), [SpuSetCommonMasterVolume\(\)](#), [SpuGetCommonMasterVolume\(\)](#), [SpuGetCommonMasterVolumeAttr\(\)](#)

SpuSetEnv

Set basic sound library environment.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuSetEnv(  
SpuEnv *env)           Basic sound library environment attribute
```

Explanation

Sets the basic sound library environment. *env.mask* contains a bit for each attribute (member of *env*). Set the bits of *env.mask* for the attribute, then set its value in *env*. When *env.mask* is 0, all attributes are set.

Currently there is only one attribute, *env.queueing*, specified by the attribute bit SPU_ENV_EVENT_QUEUEING. It establishes whether the following events are queued or not: Key On/Off, Pitch LFO Voice Setting, Noise Voice Setting, and Reverb Voice Setting. SPU_ON means the events are queued. SPU_OFF means the events are performed immediately (the default value).

See also

[SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#), [SpuSetPitchLFOVoice\(\)](#), [SpuSetNoiseVoice\(\)](#), [SpuSetReverbVoice\(\)](#), [SpuFlush\(\)](#), [SpuEnv\(\)](#)

SpuSetESA

Set starting address for straight PCM playback

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	4.5	2/24/99

Syntax

```
void SpuSetESA(  
long revAddr)
```

Starting address of SPU memory (local SPU) used for PCM playback

Explanation

SPU waveform memory can contain compressed ADPCM waveforms, but reverb memory can only contain uncompressed (straight) PCM. The SpuSetESA() function is used for playing back these straight PCM waveforms.

Note, however:

- This function cannot be used together with reverb.
- It is restricted to 22kHz, 16bit, little-endian PCM data.
- It is currently restricted to monaural, one channel.

A typical process is as follows:

1. Initialize the SPU.
2. Set reverb mode OFF.
3. Use SpuSetESA() to set the starting address of SPU memory of the PCM data to be played back (playback buffer).
4. Set the transfer address to the same starting address with SpuSetTransferStartAddr().
5. Clear the playback buffer.
6. Adjust the volume with SetSpuReverbModeDepth().
7. Use SpuWrite() to transfer PCM data from main memory to the playback buffer.

The value of *revAddr* depends on the size of the playback buffer in SPU memory that will be used for straight PCM playback. Usually the value is 0x80000 minus the buffer size.

Immediately after the data has been transferred, the buffer data will be played back repeatedly. If it is desired to play back a large amount of data that will not fit in the buffer, an SPU interrupt should be used to perform double buffering or similar processing.

The value set with SpuSetESA() is valid until the reverb mode changes with a function like SpuSetReverbModeParam(). Conversely, it is necessary to call SpuSetESA() again whenever the reverb mode changes.

See also

[SpuSetTransferStartAddr\(\)](#), [SpuWrite\(\)](#), [SpuSetReverbModeParam\(\)](#), [SpuSetReverbModeDepth\(\)](#)

SpuSetIRQ

Turn interrupt request ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

**long SpuSetIRQ(
long *on_off*)** Sets interrupt request ON/OFF/RESET

Explanation

Turns interrupt request ON/OFF.

Values of *on_off* can be:

SPU_ON	Set interrupt request
SPU_OFF	Cancel interrupt request
SPU_RESET	Reset interrupt request (cancel current interrupt request and reset)

Return value

The value that was set (SPU_ON, SPU_OFF, or SPU_RESET).

See also

[SpuGetIRQ\(\)](#)

SpuSetIRQAddr

Set interrupt request address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
u_long SpuSetIRQAddr(  
u_long addr)           Interrupt request address
```

Explanation

Sets interrupt request address value. *addr* is in bytes, and must be divisible by 8 and less than 512KB.

The interrupt request occurs when a read/write to the address takes place.

Return value

The address that was set.

If *addr* is not divisible by 8, it is increased to the next value divisible by 8, and that value is set and returned. If the address exceeds 512 KB, 0 is returned.

See also

[SpuGetIRQAddr\(\)](#), [SpuSetIRQ\(\)](#), [SpuGetIRQ\(\)](#)

SpuSetIRQCallback

Set callback for interrupt request.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

SpuSetIRQCallback(
SpuIRQCallbackProc *func*) The callback function activated at the time of an interrupt request

Explanation

Sets a callback function to be activated when an interrupt request occurs. If *func* is set to NULL, the callback is cleared.

Return value

Pointer to the previously set callback function.

See also

[SpuSetIRQ\(\)](#), [SpuSetIRQAddr\(\)](#)

SpuSetKey

Set key on/key off for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
void SpuSetKey(
long on_off,           Sets key on/key off
u_long voice_bit)     Set voice
```

Explanation

Sets key on/key off value for each voice specified by *voice_bit*. (SPU_ON = key on; SPU_OFF = key off)

Set *voice_bit* by ORing together the bits for each voice (SPU_0CH, SPU_1CH...SPU_23CH). For example, to set key on for voice 0 and voice 2:

```
SpuSetKey (SPU_ON, /* set key on */
           SPU_0CH | SPU_2CH); /* 0 ch and 2 ch */
```

See also

[SpuSetKeyOnWithAttr\(\)](#), [SpuSetVoiceAttr\(\)](#)

SpuSetKeyOnWithAttr

Set key on with attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
void SpuSetKeyOnWithAttr(  
SpuVoiceAttr *attr)           Pointer to voice attributes
```

Explanation

Specifies attributes for each voice and sets key on.

Explicitly specify the voices to be produced by ORing together the appropriate bits (SPU_0CH, SPU_1CH...SPU_23CH) in *attr.voice*.

Specify the attributes to be set by ORing together the appropriate bits in *attr.mask* and setting the corresponding values of *attr*. (If *attr.mask* is 0, all attributes are set.) See SpuSetVoiceAttr() (Table 15–6) for a list of the attributes and their descriptions.

See also

[SpuSetKey\(\)](#), [SpuSetVoiceAttr\(\)](#), [SpuGetVoiceAttr\(\)](#), [SpuVoiceAttr\(\)](#)

SpuSetMute

Turn sound muting ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuSetMute(  
long on_off)           Mute ON/OFF
```

Explanation

Turns sound muting ON/OFF. SPU_ON = Mute on; SPU_OFF = mute off.

Note: CD input and external digital input are not affected.

Return value

The value set.

See also

[SpuGetMute\(\)](#)

SpuSetNoiseClock

Set noise source clock.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuSetNoiseClock(
long n_clock)           Noise source clock
```

Explanation

Sets noise source clock to *n_clock*. The value must be in the range 0-0x3f. It is applied to all voices for whom the noise source is set with SpuSetNoiseVoice().

Return value

The noise source clock value set.

See also

[SpuGetNoiseClock\(\)](#), [SpuSetNoiseVoice\(\)](#), [SpuGetNoiseVoice\(\)](#)

SpuSetNoiseVoice

Turn noise source ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.0	12/14/98

Syntax

u_long SpuSetNoiseVoice(
long *on_off*, Sets noise source ON (SPU_ON), OFF (SPU_OFF), or direct bit pattern (SPU_BIT)
u_long *voice_bit*) Set voice

Explanation

Turns noise source on or off for specific voices. Any number of voices may be specified in *voice_bit* by setting the bit values SPU_0CH...SPU_23CH.

on_off can have the following settings:

SPU_ON	Noise source turned on for voices whose bits in <i>voice_bit</i> are 1
SPU_OFF	Noise source turned off for voices whose bits in <i>voice_bit</i> are 1
SPU_BIT	Noise source turned on for voices whose bits in <i>voice_bit</i> are 1, and turned off for voices whose bits are 0

```
    SpuSetNoiseVoice(SPU_ON,      /*set noise source on*/  
                     SPU_0CH | SPU_2CH); /*0 ch and 2 ch*/
```

Return value

An unsigned long whose low 24 bits show the current noise source on/off value for each voice (after setting). To check any voice, AND with the appropriate mask SPU_0CH...SPU_23CH.

See also

[SpuSetNoiseClock\(\)](#), [SpuGetNoiseClock\(\)](#), [SpuGetNoiseVoice\(\)](#)

SpuSetPitchLFOVoice

Set pitch LFO ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
u_long SpuSetPitchLFOVoice(  
long on_off,
```

SPU_ON: Sets pitch LFO on
SPU_OFF: Sets pitch LFO off
SPU_BIT: Sets direct bit pattern
Set voice

```
u_long voice_bit)
```

Explanation

Turns pitch LFO on or off for specific voices. Any number of voices may be specified in *voice_bit* by setting the bit values SPU_0CH...SPU_23CH.

When pitch LFO is on, voice *n* is set so that LFO sets pitch when the volume of voice (*n*-1) undergoes a time change. To allow pitch LFO, voice *n* and voice (*n*-1) must be started. Voice (*n* - 1) can produce sound at an optional timing after voice *n* starts; LFO is applied at the moment when voice (*n*-1) starts playback.

on_off can have the following settings:

SPU_ON	Pitch LFO turned on for voices whose bits in <i>voice_bit</i> are 1
SPU_OFF	Pitch LFO turned off for voices whose bits in <i>voice_bit</i> are 1
SPU_BIT	Pitch LFO turned on for voices whose bits in <i>voice_bit</i> are 1, and turned off for voices whose bits are 0

Return value

An unsigned long whose low 24 bits show the current pitch LFO on/off value for each voice (after setting). To check any voice, AND with the appropriate mask SPU_0CH...SPU_23CH.

See also

[SpuGetPitchLFOVoice\(\)](#), [SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#)

SpuSetReverb

Turn reverb ON/OFF.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuSetReverb(
long on_off)
```

SPU_ON: Set reverb on
SPU_OFF: Set reverb off

Explanation

Turns reverb on or off.

If *on_off* is SPU_OFF, if a reverb work area was not reserved with `SpuReserveReverbWorkArea()`, this function checks whether the area is available (i.e. not allocated by `SpuMalloc()`). If it is available, reverb is turned on and SPU_ON is returned. If not, reverb is turned off and SPU_OFF is returned. If it is not being

Return value

The reverb on/off value (SPU_ON or SPU_OFF)

See also

[SpuGetReverb\(\)](#), [SpuSetReverbModeParam\(\)](#), [SpuReserveReverbWorkArea\(\)](#)

SpuSetReverbDepth

Set the reverb depth parameter.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuSetReverbDepth(
    SpuReverbAttr *attr)    Pointer to reverb attribute
```

Explanation

Sets the reverb depth parameter attribute. It is set independently for left and right, by setting the appropriate bits (SPU_REV_DEPTH_L for left, SPU_REV_DEPTH_R for right) of *attr.mask*. (If *attr.mask* is 0, left and right attributes are set simultaneously.)

The range for reverb depth is -0x8000 to 0x7fff. If the value is negative, the reverb sound (wet) phase is inverted.

Return value

0.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#)

SpuSetReverbModeDelayTime

Set reverb delay time.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuSetReverbModeDelayTime(
long delay)                Reverb delay time
```

Explanation

Sets the reverb delay time, in the range 0-127. Equivalent to the SPU_REV_DELAYTIME mask setting of SpuSetReverbModeParam().

Delay time is effective only with reverb modes SPU_REV_MODE_ECHO or SPU_REV_MODE_DELAY. There is no effect if any other mode is set.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuGetReverbModeDelayTime\(\)](#)

SpuSetReverbModeDepth

Set reverb mode depth.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.7	12/14/98

Syntax

```
void SpuSetReverbModeDepth(  
short depthL,           reverb depth (left)  
short depthR)           reverb depth (right)
```

Explanation

Sets the reverb depth. Values are set independently for left and right, in the range -0x8000 to 0x7fff. If the value is negative, the reverb sound (wet) phase is inverted. Equivalent to the SPU_REV_DEPTHL and SPU_REV_DEPTHR mask settings of SpuSetReverbModeParam().

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuGetReverbModeDepth\(\)](#), [SpuSetReverbDepth\(\)](#)

SpuSetReverbModeFeedback

Set reverb feedback.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuSetReverbModeFeedback(  
long feedback)           Reverb feedback
```

Explanation

Sets the reverb feedback. Values can be 0-127. Equivalent to the SPU_REV_FEEDBACK mask setting of SpuSetReverbModeParam().

Feedback is effective only with reverb modes SPU_REV_MODE_ECHO or SPU_REV_MODE_DELAY. There is no effect with any other mode.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuGetReverbModeFeedback\(\)](#)

SpuSetReverbModeParam

Set reverb mode and attributes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
long SpuSetReverbModeParam(
    SpuReverbAttr *attr)           Pointer to reverb attributes
```

Explanation

Sets reverb mode and attributes.

You can specify the attributes (members of *attr*) to be set by ORing together the appropriate bits of *attr.mask* (see Table 15–4). If *attr.mask* is 0, all attributes are set.

Table 15–4 Reverb attributes

Attribute	Description
SPU_REV_MODE	Mode setting
SPU_REV_DEPTHL	Reverb depth (left)
SPU_REV_DEPTHR	Reverb depth (right)
SPU_REV_DELAYTIME	Delay time (ECHO, DELAY only)
SPU_REV_FEEDBACK	Feedback (ECHO, DELAY only)

- **Reverb Mode (*attr.mode*)**
When reverb mode is changed (which happens even at initial setting, because the initial value is SPU_REV_MODE_OFF), the internal reverb depth value is 0 even if depth was previously set by SpuSetReverbModeParam(). This is because the work area size changes when this mode changes, so incorrect data in the work area produces noise. So after the reverb mode changes, depth needs to be reset using SpuSetReverbModeParam() or SpuSetReverbDepth().

Based on reverb characteristics, the time to complete one scan of the work area is estimated and the mode/depth are set; or, after the mode is set, the work area data is erased, then depth is set.

The size the work area depends on the reverb mode as shown below. However, this area is managed by a memory management mechanism such as SpuMalloc(). See SpuMalloc() for details.

Table 15–5: Sound Buffer Work Area Size for Reverb Modes

<i>attr.mode</i>	mode	hexadecimal	decimal
SPU_REV_MODE_OFF	off	0/80*	0/128*
SPU_REV_MODE_ROOM	room	26c0	9920
SPU_REV_MODE_STUDIO_A	studio (small)	1f40	8000
SPU_REV_MODE_STUDIO_B	studio (med)	4840	18496
SPU_REV_MODE_STUDIO_C	studio (big)	6fe0	28640
SPU_REV_MODE_HALL	hall	ade0	44512
SPU_REV_MODE_SPACE	space echo	f6c0	63168
SPU_REV_MODE_ECHO	echo	18040	98368
SPU_REV_MODE_DELAY	delay	18040	98368
SPU_REV_MODE_PIPE	half echo	3c00	15360

***Note:** 128 bytes if SpuReserveReverbWorkArea (SPU_ON) is used for address setting, even if the mode is off; 0 bytes otherwise.

If SPU_REV_MODE_CLEAR_WA is set in *attr.mode*, the reverb work area is cleared, as a measure against noise when changing modes. Since the sound buffer is cleared by synchronous DMA transfer, other processing (drawing, sound generation) is blocked during this process.

SpuClearReverbWorkArea() can also be used to clear the work area.

- **Reverb Depth** (*attr.depth*)
Values are set independently for left and right, in the range -0x8000 to 0x7fff. If the value is negative, the reverb sound (wet) phase is inverted.
- **Delay Time** (*attr.delay*)
Values are in the range 0-127. Valid only when mode is SPU_REV_MODE_ECHO or SPU_REV_MODE_DELAY.
- **Feedback** (*attr.feedback*)
Values are from 0 to 127. Valid only when mode is SPU_REV_MODE_ECHO or SPU_REV_MODE_DELAY.

Return value

If the area needed as a work area by the new mode was allocated for another area `SpuMalloc()`/`SpuMallocWithStartAddr()`, none of the set reverb attributes are set and `SPU_ERROR` is returned. If it is not being used, the set reverb attributes are set and 0 is returned.

`SPU_ERROR` is also returned when an invalid `SPU_REV_MODE` is set.

See also

[SpuGetReverbModeParam\(\)](#), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#), [SpuReserveReverbWorkArea\(\)](#), [SpuClearReverbWorkArea\(\)](#)

SpuSetReverbModeType

Set reverb mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.7	12/14/98

Syntax

```
void SpuSetReverbModeType(
long type)                Reverb mode type
```

Explanation

Sets the reverb mode. Equivalent to the SPU_REV_MODE mask setting of SpuSetReverbModeParam().

See Table 15–5 under SpuSetReverbModeParam() for the possible values of *type*.

See also

[SpuSetReverbModeParam\(\)](#), [SpuGetReverbModeParam\(\)](#), [SpuMalloc\(\)](#), [SpuMallocWithStartAddr\(\)](#), [SpuReserveReverbWorkArea\(\)](#), [SpuClearReverbWorkArea\(\)](#), [SpuGetReverbModeType\(\)](#)

SpuSetReverbVoice

Set reverb ON/OFF for each voice.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.0	12/14/98

Syntax

u_long **SpuSetReverbVoice**(
long *on_off*,
u_long *voice_bit*)

Sets reverb ON (SPU_ON), OFF (SPU_OFF), or direct bit pattern (SPU_BIT)
Set voice

Explanation

Turns reverb on or off for specific voices. Any number of voices may be specified in *voice_bit* by setting the bit values SPU_0CH...SPU_23CH.

on_off can have the following settings:

- SPU_ON Reverb turned on for voices whose bits in *voice_bit* are 1
- SPU_OFF Reverb turned off for voices whose bits in *voice_bit* are 1
- SPU_BIT Reverb turned on for voices whose bits in *voice_bit* are 1, and turned off for voices whose bits are 0

For example, to set voice 0 and voice 2 reverb on:

```
    SpuSetReverbVoice(SPU_ON,    /*set reverb on*/  
        SPU_0CH | SPU_2CH);    /*0 ch and 2 ch*/
```

Return value

An unsigned long whose low 24 bits show the current noise source on/off value for each voice (after setting). To check any voice, AND with the appropriate mask SPU_0CH...SPU_23CH.

See also

[SpuGetReverbVoice\(\)](#)

SpuSetTransferCallback

Set callback function for completion of DMA transfer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.1	12/14/98

Structure

SpuTransferCallbackProc **SpuSetTransferCallback**(
SpuTransferCallbackProc *func*) Callback function for completion of DMA transfer.

Explanation

Sets function to be called when DMA transfer is completed. If *func* is NULL, the callback is cleared.

When a callback set by this function executes at DMA transfer completion, `SpulsTransferCompleted()` does not function.

Return value

The previously set callback function. If no callback function was set, NULL is returned.

See also

[SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#), [SpuRead\(\)](#), [SpuReadDecodedData\(\)](#), [SpuSetTransferMode\(\)](#), [SpuGetTransferMode\(\)](#), [SpulsTransferCompleted\(\)](#)

SpuSetTransferMode

Set sound buffer transfer mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

long SpuSetTransferMode(
long mode) Mode: see table below

Explanation

Sets the mode for transferring data from main memory to the sound buffer. The mode values can be:

- SPU_TRANSFER_BY_DMA: DMA transfer; can do other processing during transfer (default value).
- SPU_TRANSFER_BY_IO: I/O transfer. Uses CPU; cannot do other processing during transfer.

Note: These specifications are valid only when transferring data from main memory to the sound buffer. DMA transfer is always used when transferring data from the sound buffer to main memory.

When a transfer is done without first calling this function, the transfer mode is the previously set value.

Return value

The transfer mode set (SPU_TRANSFER_BY_DMA or SPU_TRANSFER_BY_IO)

See also

[SpuGetTransferMode\(\)](#), [SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#).

SpuSetTransferStartAddr

Set sound buffer transfer start address.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

u_long SpuSetTransferStartAddr(
u_long *addr*) Sound buffer transfer destination/transfer source start address

Explanation

Sets a starting address in the sound buffer, specified in *addr*, for transferring data to and from main memory. *addr* must be a byte value that is

- Divisible by 8. If it is not divisible by 8, it is increased to the next value divisible by 8.
- Between 0x1010 - 0x7fff for transfers to the sound buffer.
- Between 0 - 0x0fff for transfers from the sound buffer. See SpuReadDecodedData().

Note: 0x1000 - 0x100f is reserved for the system.

Return value

Start address value. If the address specified is smaller than 0x1010 or greater than 512 KB, 0 is returned.

See also

[SpuGetTransferStartAddr\(\)](#), [SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuWritePartly\(\)](#), [SpuRead\(\)](#), [SpuReadDecodedData\(\)](#)

SpuSetVoiceADSR

Set ADSR values.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.6	12/14/98

Syntax

```
void SpuSetVoiceADSR(  
int voiceNum,           Voice number (0 - 23)  
u_short AR,             ADSR attack rate  
u_short DR,             ADSR decay rate  
u_short SR,             ADSR sustain rate  
u_short RR,             ADSR release rate  
u_short SL)             ADSR sustain level
```

Explanation

Sets individual ADSR attributes for a voice. Corresponds to SpuSetVoiceAttr() mask specifications SPU_VOICE_ADSR_AR / SPU_VOICE_ADSR_DR / SPU_VOICE_ADSR_SR / SPU_VOICE_ADSR_RR / SPU_VOICE_ADSR_SL.

The rate modes used are:

- Attack Rate: SPU_VOICE_LINEARIncN (Linear Increase)
- Sustain Rate: SPU_VOICE_LINEARDecN (Linear Decrease)
- Release Rate: SPU_VOICE_LINEARDecN (Linear Decrease)

See Table 15–10 under SpuSetVoiceAttr() for values that can be specified for each rate. To set multiple rate modes at the same time, use SpuSetVoiceADSRAttr().

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceAR\(\)](#), [SpuSetVoiceDR\(\)](#), [SpuSetVoiceSR\(\)](#), [SpuSetVoiceRR\(\)](#), [SpuSetVoiceSL\(\)](#)

SpuSetVoiceADSRAttr

Set ADSR and ADSR modes.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	8/9/99

Syntax

```
void SpuSetVoiceADSRAttr(
  int voiceNum,           Voice number (0 - 23)
  u_short AR,             ADSR attack rate
  u_short DR,             ADSR decay rate
  u_short SR,             ADSR sustain rate
  u_short RR,             ADSR release rate
  u_short SL,             ADSR sustain level
  long ARmode,            ADSR attack rate mode
  long SRmode,            ADSR sustain rate mode
  long RRmode)            ADSR release rate mode
```

Explanation

Sets ADSR attributes and mode. Corresponds to SpuSetVoiceAttr() mask specifications
 SPU_VOICE_ADSR_AR / SPU_VOICE_ADSR_DR / SPU_VOICE_ADSR_SR / SPU_VOICE_ADSR_RR /
 SPU_VOICE_ADSR_SL / SPU_VOICE_ADSR_AMODE / SPU_VOICE_ADSR_SMODE /
 SPU_VOICE_ADSR_RMODE.

Refer to SpuSetVoiceAttr() for values that can be specified in each rate and rate mode.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceADSR\(\)](#), [SpuSetVoiceAR\(\)](#), [SpuSetVoiceDR\(\)](#),
[SpuSetVoiceSR\(\)](#), [SpuSetVoiceRR\(\)](#), [SpuSetVoiceSL\(\)](#), [SpuSetVoiceARAttr\(\)](#), [SpuSetVoiceSRAttr\(\)](#),
[SpuSetVoiceRRAttr\(\)](#)

SpuSetVoiceAR

Set ADSR attack rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceAR(

int *voiceNum*, Voice number (0 - 23)

u_short *AR*) ADSR attack rate

Explanation

Sets ADSR attack rate for a voice. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_ADSR_AR.

ADSR attack rate mode becomes SPU_VOICE_LINEARIncN (Linear increase mode). To set ADSR attack rate and ADSR attack rate mode at the same time, use SpuSetVoiceARAttr().

Refer to SpuSetVoiceAttr() for values that can be specified in *AR*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceARAttr\(\)](#)

SpuSetVoiceARAttr

Set ADSR attack rate / attack rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceARAttr(

int *voiceNum*, Voice number (0 - 23)
u_short *AR*, ADSR attack rate
long *ARmode*) ADSR attack rate mode

Explanation

Sets ADSR attack rate / ADSR attack rate mode for a voice. Corresponds to SpuSetVoiceAttr() mask specifications SPU_VOICE_ADSR_AR and SPU_VOICE_ADSR_AMODE. Refer to SpuSetVoiceAttr() for values that can be specified in *AR* and *ARmode*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceAR\(\)](#)

SpuSetVoiceAttr

Set attributes for each voice.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
void SpuSetVoiceAttr(
    SpuVoiceAttr *attr)    Pointer to voice attributes
```

Explanation

Sets attributes for one or more voices.

To specify the voices whose attributes you wish to set, OR together the appropriate bits of *attr.voice* (SPU_0CH...SPU_23CH).

To specify which attributes to set, OR together the terms shown below in *attr.mask*, then set the values of the corresponding members of *attr*. (If *attr.mask* is 0, all attributes are set.)

Table 15–6 Voice Attributes

Attribute value for <i>attr.mask</i>	Description	Member of <i>attr</i> to set
SPU_VOICE_VOLL	Volume (left)	<i>volume</i>
SPU_VOICE_VOLR	Volume (right)	<i>volume</i>
SPU_VOICE_VOLMODEL	Volume mode (left)	<i>volmode</i>
SPU_VOICE_VOLMODER	Volume mode (right)	<i>volmode</i>
SPU_VOICE_PITCH	Interval (pitch specification)	<i>pitch</i>
SPU_VOICE_NOTE	Interval (note specification)	<i>note</i>
SPU_VOICE_SAMPLE_NOTE	Waveform data sample note	<i>sample_note</i>
SPU_VOICE_WDSA	Waveform data start address	<i>addr</i>
SPU_VOICE_ADSR_AMODE	ADSR Attack rate mode	<i>a_mode</i>
SPU_VOICE_ADSR_SMODE	ADSR Sustain rate mode	<i>s_mode</i>
SPU_VOICE_ADSR_RMODE	ADSR Release rate mode	<i>r_mode</i>
SPU_VOICE_ADSR_AR	ADSR Attack rate	<i>ar</i>
SPU_VOICE_ADSR_DR	ADSR Decay rate	<i>dr</i>
SPU_VOICE_ADSR_SR	ADSR Sustain rate	<i>sr</i>
SPU_VOICE_ADSR_RR	ADSR Release rate	<i>rr</i>
SPU_VOICE_ADSR_SL	ADSR Sustain level	<i>sl</i>
SPU_VOICE_ADSR_ADSR1	ADSR adsr1 for 'VagAtr'	<i>adsr1</i>
SPU_VOICE_ADSR_ADSR2	ADSR adsr2 for 'VagAtr'	<i>adsr2</i>
SPU_VOICE_LSAX	Loop start address	<i>loop_addr</i>

The individual settings are described below.

- Volume and Volume Mode
The volume modes and their range of possible volume settings are shown below:

Table 15–7: Volume Mode and Volume Setting Ranges

Mode (phase)	SPU_VOICE_VOLMODEx	SPU_VOICE_VOLx
Direct mode	SPU_VOICE_DIRECT	-0x4000 - 0x3fff
Linear inc. mode	SPU_VOICE_LINEARIncN	0x00 - 0x7f (normal)
Linear inc. mode	SPU_VOICE_LINEARIncR	0x00 - 0x7f (inverted)
Linear dec. mode	SPU_VOICE_LINEARDecN	0x00 - 0x7f (normal)
Linear dec. mode	SPU_VOICE_LINEARDecR	0x00 - 0x7f (inverted)
Expon. inc. mode	SPU_VOICE_EXPIncN	0x00 - 0x7f (normal)
Expon. inc. mode	SPU_VOICE_EXPIncR	0x00 - 0x7f (inverted)
Expon. dec. mode	SPU_VOICE_EXPDec	0x00 - 0x7f

1. Direct Mode
Specifies a fixed volume (the default mode). When the volume is negative, its phase is inverted.
 2. Linear Increase Mode (Normal Phase)
When the current volume value is positive, volume increases linearly from the current value to the maximum value.
 3. Linear Increase Mode (Inverted Phase)
When the current volume value is negative (inverted phase), volume increases linearly from the current value to the maximum value, with phase inverted.
 4. Linear Decrease Mode (Normal Phase)
When the current volume value is positive, volume decreases linearly from the current value to the minimum volume value.
 5. Linear Decrease Mode (Inverted Phase)
When the current volume value is negative (inverted phase), volume decreases linearly from the current value to the minimum volume value, with phase inverted.
 6. Exponential Increase Mode (Normal Phase)
When the current volume value is positive, volume increases exponentially from the current value to the maximum value.
 7. Exponential Increase Mode (Inverted Phase)
When the current volume value is negative (inverted phase), volume increases exponentially from the current value to the maximum value, with phase inverted.
 8. Exponential Decrease Mode
Whether the current volume value is positive or negative, volume decreases exponentially from the current value to the minimum volume value.
- Playback rate (set pitch, set note)
May be set by the two methods listed below:
 1. Pitch specification: specifies an interval in *attr.pitch* in the range 0x0000-0x3fff.

Table 15–8: Pitch Specification Values and Interval

Value Set	0x0200	0x0400	0x0800	0x1000	0x2000	0x3fff
Interval	- 3 oct.	- 2 oct.	- 1 oct.	tone	+ 1 oct.	+ 2 oct.

2. Note specification: specifies an interval in *attr.note* as follows, using a 16-bit value for note and cent (here, the value of a half tone divided by 128).

Table 15–9: Note/Sample Note Specification Values

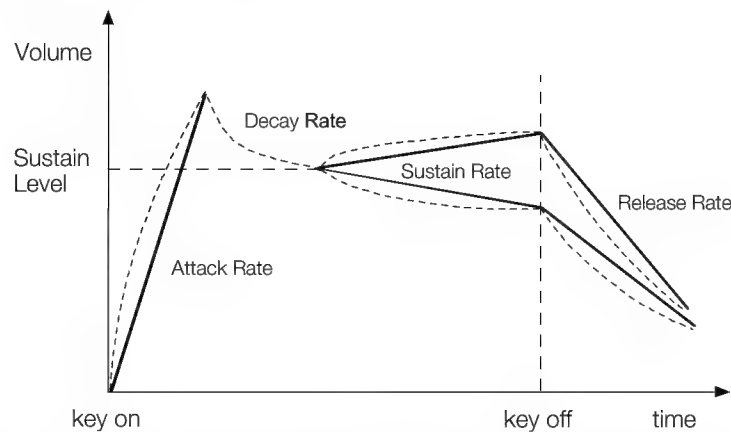
Bit	Value Set
Upper 8 bits	MIDI note number
Lower 8 bits	Cent (expressed as a half tone divided by 128)

This setting cannot be used unless the waveform data sample note feature, described below, is set.

- Waveform Data Sample Note
Sets interval in *attr.sample_note* at the time of sampling, using a 16-bit value for note and cent, as shown in Table 15–9. Setting this value makes it possible to set the playback rate as above.
- Waveform Data Start Address
attr.add specifies the sound buffer starting address of the waveform data you want to produce in the voice.
- Loop Start Address
If waveform data that generates sound in a voice is created with a loop specified, and if the waveform starting address is set, it is unnecessary to set the loop start address explicitly.
However, when you wish to set a loop start address dynamically at the time of execution, you must set *attr.loop_addr* to the start address of the loop in the sound buffer.
If a loop was not set at the time of waveform data creation, even if *SPU_VOICE_LSAX* is specified and set in *attr.loop_addr*, that setting is invalid.

- ADSR
A conceptual diagram of ADSR is shown below.

Figure 15-1: ADSR Conceptual Diagram



The attributes that can be set and their ranges are shown in Table 15-10.

Table 15-10: Rate and Level Setting Ranges

Attribute	Structure Member	Setting Range
Attack rate	<i>attr.ar</i>	0x00 - 0x7f
Decay rate	<i>attr.dr</i>	0x0 - 0xf
Sustain rate	<i>attr.sr</i>	0x00 - 0x7f
Release rate	<i>attr.rr</i>	0x00 - 0x1f
Sustain level	<i>attr.sl</i>	0x0 - 0xf

Rate curves may be set for Attack, Sustain, Release (see Table 15-11). Because only exponential decrease may be used for Decay, that attribute cannot be set.

Table 15-11: ADSR Rate Modes

Attribute	Settable modes
Attack rate	SPU_VOICE_LINEARIncN (linear increase)
(<i>attr.a_mode</i>)	SPU_VOICE_EXPIncN (exponential increase)
Sustain rate	SPU_VOICE_LINEARIncN (linear increase)
(<i>attr.s_mode</i>)	SPU_VOICE_LINEARDecN (linear decrease)
	SPU_VOICE_EXPIncN (exponential increase)
	SPU_VOICE_EXPDec (exponential decrease)
Release rate	SPU_VOICE_LINEARDecN (linear decrease)
(<i>attr.r_mode</i>)	SPU_VOICE_EXPDec (exponential decrease)

Also, data from structure VagAtr members *adsr1* and *adsr2* may be set directly in *attr.adsr1* and *attr.adsr2*. In this case only SPU_VOICE_ADSR_ADSR1 and SPU_VOICE_ADSR_ADSR2 can be set for ADSR in *attr.mask*.

See also

[SpuRSetVoiceAttr\(\)](#), [SpuGetVoiceAttr\(\)](#), [SpuSetKey\(\)](#), [SpuSetKeyOnWithAttr\(\)](#), [SpuVoiceAttr\(\)](#).

SpuSetVoiceDR

Set ADSR decay rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceDR(

int *voiceNum*, Voice number (0 - 23)
u_short *DR*) ADSR decay rate

Explanation

Sets ADSR decay rate used in voice *voicenum*. Corresponds to `SpuSetVoiceAttr()` mask specification `SPU_VOICE_ADSR_DR`. Refer to `SpuSetVoiceAttr()` for values that can be specified in *DR*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#)

SpuSetVoiceLoopStartAddr

Set loop start address of waveform data in sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuSetVoiceLoopStartAddr(
  int voiceNum,           Voice number (0 - 23)
  u_long loopStartAddr)   Loop start address
```

Explanation

Sets start address of waveform data in the sound buffer. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_LSAX. See SpuSetVoiceAttr() for values that can be specified in *loopStartAddr*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetTransferStartAddr\(\)](#).

SpuSetVoiceNote

Set interval (note specification).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuSetVoiceNote(
int voiceNum,      Voice number (0 - 23)
u_short note)      Interval (note specification)
```

Explanation

Sets the voice interval by note. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_NOTE.

Refer to SpuSetVoiceAttr() for values that can be specified in the interval by note specification.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceSampleNote\(\)](#)

SpuSetVoicePitch

Set interval (pitch specification).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoicePitch(

int *voiceNum*,

u_short *pitch*)

Voice number (0 - 23)

Interval (pitch specification)

Explanation

Sets the voice interval by pitch. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_PITCH. Refer to SpuSetVoiceAttr() for values that can be specified in the interval by pitch specification.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#)

SpuSetVoiceRR

Set ADSR release rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceRR(

int *voiceNum*, Voice number (0 - 23)

u_short *RR*) ADSR release rate

Explanation

Sets ADSR release rate for voice *voiceNum*. Corresponds to `SpuSetVoiceAttr()` mask specification `SPU_VOICE_ADSR_RR`.

ADSR release rate mode becomes `SPU_VOICE_LINEARDecN` (Linear decrease mode). To set release rate and release rate mode at the same time, use `SpuSetVoiceRRAttr`.

Refer to `SpuSetVoiceAttr()` for values that can be specified in *RR*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceRRAttr\(\)](#)

SpuSetVoiceRRAttr

Set ADSR release rate / release rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceRRAttr(

int *voiceNum*, Voice number (0 - 23)

u_short *RR*, ADSR release rate

long *RRmode*) ADSR release rate mode

Explanation

Sets ADSR release rate / ADSR release rate mode for voice *voiceNum*. Corresponds to SpuSetVoiceAttr() mask specifications SPU_VOICE_ADSR_RR and SPU_VOICE_ADSR_RRMODE.

Refer to SpuSetVoiceAttr() for values that can be specified in *RR* and *RRmode*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceRR\(\)](#)

SpuSetVoiceSampleNote

Set waveform data sample note.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceSampleNote(

int *voiceNum*, Voice number (0 - 23)

u_short *sampleNote*) Sets waveform data sample note

Explanation

Sets the waveform data sample note for voice *voiceNum*. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_SAMPLE_NOTE. Refer to SpuSetVoiceAttr() for values that can be specified in *sampleNote*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceNote\(\)](#)

SpuSetVoiceSL

Set ADSR sustain level.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuSetVoiceSL(
  int voiceNum,      Voice number (0 - 23)
  u_short SL)        ADSR sustain level
```

Explanation

Sets ADSR sustain level used for voice *voiceNum*. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_ADSR_SL.

ADSR sustain level mode becomes SPU_VOICE_LINEARDecN (Linear decrease mode). To set ADSR sustain level and ADSR sustain level mode at the same time, use SpuSetVoiceAttr().

Refer to SpuSetVoiceAttr() for values that can be specified in *SL*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceRRAttr\(\)](#)

SpuSetVoiceSR

Set ADSR sustain rate.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuSetVoiceSR(
  int voiceNum,      Voice number (0 - 23)
  u_short SR)        ADSR sustain rate
```

Explanation

Sets ADSR sustain rate used for voice *voiceNum*. Corresponds to `SpuSetVoiceAttr()` mask specification `SPU_VOICE_ADSR_SR`.

ADSR sustain rate mode becomes `SPU_VOICE_LINEARDecN` (Linear decrease mode). To set ADSR sustain rate and ADSR sustain rate mode at the same time, use `SpuSetVoiceSRAttr()`.

Refer to `SpuSetVoiceAttr()` for values that can be specified in *SR*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceSRAttr\(\)](#)

SpuSetVoiceSRAttr

Set ADSR sustain rate / sustain rate mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceSRAttr(

int *voiceNum*, Voice number (0 - 23)

u_short *SR*, ADSR sustain rate

long *SRmode*) ADSR sustain rate mode

Explanation

Sets ADSR sustain rate / ADSR sustain rate mode used for voice *voiceNum*. Corresponds to SpuSetVoiceAttr() mask specifications SPU_VOICE_ADSR_SR and SPU_VOICE_ADSR_SRMODE. Refer to SpuSetVoiceAttr() for values that can be specified in *SR* and *SRmode*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceSR\(\)](#)

SpuSetVoiceStartAddr

Set start address of waveform data in sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

```
void SpuSetVoiceStartAddr(
int voiceNum,           Voice number (0 - 23)
u_long startAddr)       Waveform data start address
```

Explanation

Sets start address of waveform data in the sound buffer. Corresponds to SpuSetVoiceAttr() mask specification SPU_VOICE_WDSA. See SpuSetTransferStartAddr() for allowable values of *startAddr*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetTransferStartAddr\(\)](#)

SpuSetVoiceVolume

Set voice volume.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceVolume(

int *voiceNum*, Voice Number (0 - 23)

short *volumeL*, Volume (Left)

short *volumeR*) Volume (Right)

Explanation

Sets the voice volume. Corresponds to SpuSetVoiceAttr() mask specifications SPU_VOICE_VOLL and SPU_VOICE_VOLR.

Volume mode becomes "Direct Mode", and the range of values that can be specified in *volumeL* and *volumeR* is equivalent to "Direct Mode" of SpuSetVoiceAttr(). To specify both volume and volume mode at the same time, use SpuSetVoiceVolumeAttr(). See SpuSetVoiceAttr() for values that can be specified in *volumeL* and/or *volumeR*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceVolumeAttr\(\)](#)

SpuSetVoiceVolumeAttr

Set voice volume/volume mode.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.6	12/14/98

Syntax

void SpuSetVoiceVolumeAttr(

int <i>voiceNum</i> ,	Voice Number (0 - 23)
short <i>volumeL</i> ,	Volume (Left)
short <i>volumeR</i> ,	Volume (Right)
short <i>volModeL</i> ,	Volume mode (Left)
short <i>volModeR</i>)	Volume mode (Right)

Explanation

Sets voice volume and/or volume mode. Corresponds to SpuSetVoiceAttr() mask specifications SPU_VOICE_VOLL / SPU_VOICE_VOLR / SPU_VOICE_VOLMODEL / SPU_VOICE_VOLMODER.

See SpuSetVoiceAttr() for values that can be specified in *volModeL*, *volModeR*, *volumeL* and/or *volumeR*.

See also

[SpuSetVoiceAttr\(\)](#), [SpuNSetVoiceAttr\(\)](#), [SpuSetVoiceVolumeAttr\(\)](#)

SpuStart

Start SPU processing.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

void SpuStart(void)

Explanation

Starts SPU processing. This function is also called by Spulnit(), so it is not necessary to call it when initializing, but SpuStart() must be called after calling SpuQuit() if you use SpuQuit() to turn functionality off.

In the current specification, DMA transfer initialization is done after SpuStart() is called.

See also

[SpuQuit\(\)](#), [Spulnit\(\)](#).

SpuStGetStatus

Get SPU streaming status.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

long SpuStGetStatus(void)

Explanation

Determines the state of SPU streaming.

Return value

Table 15–12 SPU streaming status

Attribute	Description
SPU_ST_NOT_AVAILABLE	SPU streaming is not available; SpuStInit() has not been called.
SPU_ST_IDLE	Data transfer to the sound buffer has not been performed yet or all streams have terminated.
SPU_ST_PREPARE	Transferring the first buffer.
SPU_ST_TRANSFER	Transferring the data to the sound buffer. If SpuStTransfer (SPU_ST_PREPARE) is executed for a voice in this state, status does not change to SPU_ST_PREPARE.
SPU_ST_FINAL	Waiting for the end of playback after transferring the last buffer. SpuStTransfer() is not accepted in this state.

See also

[SpuStInit\(\)](#), [SpuStTransfer\(\)](#), [SpuStGetVoiceStatus\(\)](#)

SpuStGetVoiceStatus

Determine voices used for SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

u_long SpuStGetVoiceStatus(void)

Explanation

Determines the voices used for SPU streaming.

Return Value

Value of the voices represented by the bit OR of SPU_0CH ... SPU_23CH.

See also

[SpuStTransfer\(\)](#), [SpuStGetStatus\(\)](#)

SpuStInit

Initialize SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

[SpuStEnv](#) *SpuStInit(
long *mode*) Not used under the current specification. Pass "0".

Explanation

Initializes SPU streaming. Called only once in an executed program.

Return Value

Pointer to the SPU streaming environment structure SpuStEnv.

See also

[SpuStQuit\(\)](#), [SpuStEnv\(\)](#)

SpuStQuit

Complete SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

long SpuStQuit(void)

Explanation

Completes SPU streaming. Prior to calling this function, processing must be completed for all the streams.

Return value

SPU_ST_ACCEPT	Normal end
SPU_ST_WRONG_STATUS	SpuStQuit() not accepted because current status is not SPU_ST_IDLE.

See also

[SpuStInit\(\)](#), [SpuStGetStatus\(\)](#)

SpuStSetPreparationFinishedCallback

Set function to be called at end of preparation phase of SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

SpuStCallbackProc

SpuStSetPreparationFinishedCallback(

SpuStCallbackProc *callback_proc*) Pointer to callback function

SpuStCallbackProc *callback_proc*(

u_long *voice_bit*,

long *status*)

Explanation

Sets the callback function to be activated at the end of the preparation state of data transfer in SPU streaming.

When *callback_proc* is called, it is passed the following arguments:

- *voice_bit* specifies the voices for whom preparation transfer has completed. You can check the voices by using the bit values SPU_0CH to SPU_23CH.
- *status* can be either SPU_ST_PREPARE (streaming is in preparation state) or SPU_ST_PLAY (playing)

Return Value

Pointer to the previously set callback function; NULL if no callback function was previously set.

See also

[SpuStTransfer\(\)](#), [SpuStSetTransferFinishedCallback\(\)](#), [SpuStSetStreamFinishedCallback\(\)](#)

SpuStSetStreamFinishedCallback

Set function to be called at completion of each stream in SPU streaming.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

SpuStCallbackProc

SpuStSetStreamFinishedCallback(

SpuStCallbackProc *callback_proc*) Pointer to callback function

SpuStCallbackProc **callback_proc*(

u_long *voice_bit*,

long *status*)

Explanation

Sets the callback function called at the completion of each stream in the SPU streaming.

When *callback_proc* is called, it is passed the following arguments:

- *voice_bit* specifies the voices for whom preparation transfer has completed. You can check the voices by using the bit values SPU_0CH to SPU_23CH.
- *status* can be either SPU_ST_FINAL (streaming is in termination state) or SPU_ST_PLAY (playing)

Return Value

Pointer to the previously set callback function; NULL if no callback function was previously set.

See also

[SpuStTransfer\(\)](#), [SpuStSetPreparationFinishedCallback\(\)](#), [SpuStSetTransferFinishedCallback\(\)](#)

SpuStSetTransferFinishedCallback

Set function to be called at completion of one transfer to the stream buffer for all streams

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.2	12/14/98

Syntax

SpuStCallbackProc

SpuStSetTransferFinishedCallback(

SpuStCallbackProc *callback_proc*) Pointer to callback function

SpuStCallbackProc **callback_proc*(

u_long *voice_bit*,

long *status*)

Explanation

Sets the callback function to be called at the completion of one transfer to the stream buffer for all the streams in the SPU streaming.

When *callback_proc* is called, it is passed the following arguments:

- *voice_bit* specifies the voices for whom transfer has completed. You can check the voices by using the bit values SPU_0CH to SPU_23CH.
- *status* is always SPU_ST_PLAY (playing).

Return Value

Pointer to the previously set callback function; NULL if no callback function was previously set.

See also

[SpuStTransfer\(\)](#), [SpuStSetPreparationFinishedCallback\(\)](#), [SpuStSetStreamFinishedCallback\(\)](#)

SpuStTransfer

Prepare for a stream and provide instructions for starting it.

Library	Header File	Introduced	Documentation Date
libspu.lib	libspu.h	3.2	12/14/98

Syntax

```
long SpuStTransfer(  
long flag,           Stream state flag  
u_long voice_bit)   Streaming voices
```

Explanation

Prepares for a stream in SPU streaming, and provides instructions for starting it.
The voices for the stream are set in *voice_bit* by ORing the appropriate values SPU_0CH ... SPU_23CH.
flag values are:

- SPU_ST_PREPARE = Preparation
Prepares the stream according to the attributes of the SpuStEnv structure returned by SpuStInit(). After preparation, the callback function set by SpuStSetPreparationFinishedCallback() is called.
- SPU_ST_PLAY = Start
The stream is started according to the attributes of the SpuStEnv structure returned by SpuStInit(). If streaming status is SPU_ST_PREPARE, the voice is keyed on. If the status is SPU_ST_TRANSFER, the transfer waits until processing for the current streams is transferred to the latter part of the stream buffer.
When one transfer to the stream buffer for all streams is completed, the callback function set by SpuStSetTransferFinishedCallback() is called, and the attributes for the next transfer for each stream are set.
When a stream is completed, the callback function set by SpuStSetStreamFinishedCallback() is called (just before the next transfer if other streams are processed.)

Return value

SPU_ST_ACCEPT	Processing is accepted.
SPU_ST_NOT_AVAILABLE	SPU streaming is not available. SpuStInit() has not been called.
SPU_ST_INVALID_ARGUMENTS	The value of the arguments is not in the specification.
SPU_ST_WRONG_STATUS	SpuStTransfer() not accepted. The causes are:

- The current status is SPU_ST_FINAL.
- *flag* is SPU_ST_PREPARE, and the current status is SPU_ST_PREPARE.
- *flag* is SPU_ST_PLAY, and the current status is SPU_ST_IDLE.

See also

[SpuStInit\(\)](#), [SpuStSetPreparationFinishedCallback\(\)](#), [SpuStSetTransferFinishedCallback\(\)](#), [SpuStSetStreamFinishedCallback\(\)](#)

SpuWrite

Transfer data from main memory to the sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
u_long SpuWrite(
u_char *addr,           Pointer to transfer data start address in main memory
u_long size)           Transfer data size (in bytes)
```

Explanation

Transfers *size* bytes of data from main memory *addr* to the sound buffer

The main memory address *addr* storing the transfer data must be a global variable or an address in a heap area that was allocated by a function such as `malloc()`. It can't address a variable on the stack declared in a function.

`SpuWrite()` does not perform sound buffer memory management, so real waveform data cannot be used if the user does not transfer to addresses which avoid the following areas.

- SPU decoded data transfer area: 0x0000-0xfff
- System reserved area: 0x1000-0x100f
- Addresses after the reverb work area offset (start) address

After calling, either call `SpulsTransferCompleted()` to confirm transfer completion or set the DMA transfer completion Callback function in advance using `SpuSetTransferCallback()`.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 is transferred, it's possible to damage the data in the SPU memory.

Return value

Transferred data size. If *size* is larger than 512 KB, the actual transferred size is returned.

If the transfer mode is `SPU_TRANSFER_BY_DMA` and *size* is not a multiple of 64, the return value will be incorrect.

See also

[SpuWrite0\(\)](#), [SpuWritePartly\(\)](#), [SpuRead\(\)](#), [SpuSetTransferMode\(\)](#), [SpuGetTransferMode\(\)](#), [SpuSetTransferStartAddr\(\)](#), [SpuGetTransferStartAddr\(\)](#), [SpulsTransferCompleted\(\)](#), [SpuSetTransferCallback\(\)](#)

SpuWrite0

Clear sound buffer.

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
u_long SpuWrite0(
u_long size)          Clear area size (in bytes)
```

Explanation

Writes 0s in the sound buffer area starting at the address specified by `SpuSetTransferStartAddr()`. The number of bytes written is *size*. The writing is done by DMA transfer, but is started synchronously.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 is transferred, it's possible to damage the data in the SPU memory.

Return value

The size of the area cleared. If *size* is larger than 512 KB, the actual written size is returned.

See also

[SpuWrite\(\)](#), [SpuWritePartly\(\)](#), [SpuRead\(\)](#), [SpuSetTransferMode\(\)](#), [SpuGetTransferMode\(\)](#), [SpuSetTransferStartAddr\(\)](#), [SpuGetTransferStartAddr\(\)](#)

SpuWritePartly

Transfer data from main memory to sound buffer (assuming the transfer is divided into sections).

Library	Header File	Introduced	Documentation Date
<i>libspu.lib</i>	<i>libspu.h</i>	3.0	12/14/98

Syntax

```
u_long SpuWritePartly(
u_char *addr,           Pointer to transfer data start address in main memory
u_long size)           Transfer data size (in bytes)
```

Explanation

Transfers data from main memory to the sound buffer.

The main memory address holding the transfer data must be a global variable or a variable in a heap area allocated by a function such as `malloc()`. It can't be a stack variable declared in a function.

Data is transferred from the address specified in `SpuSetTransferStartAddr()`, and after completion of the transfer specified by *size*, the starting address is incremented by *size*, and stored internally.

In the case of continuous transfer, the size of each transfer must be divisible by 8, except for the final block.

If `SpuSetTransferStartAddr()` is called during continuous transfer processing, correct continuous transfer is not guaranteed.

`SpuWritePartly()` does not perform sound buffer memory management, so real waveform data cannot be used if the user does not transfer to addresses which avoid the following areas.

- SPU decoded data transfer area: 0x0000-0xffff
- System reserved area: 0x1000-0x100f
- Addresses after the reverb work area offset (start) address

After calling, either call `SpulsTransferCompleted()` to confirm transfer completion or set the DMA transfer completion Callback function in advance using `SpuSetTransferCallback`.

Due to the limitations of the DMA transfer hardware, transfers are always performed in 64 byte units. When specifying values which are not multiples of 64 as secondary arguments, since the portion of the value which is a multiple of 64 is transferred, it's possible to damage the data in the SPU memory.

Return value

Transferred data size. If *size* is larger than 512 KB, the actual transferred size is returned.

If the transfer mode is `SPU_TRANSFER_BY_DMA` and *size* is not a multiple of 64, the return value will be incorrect.

See also

[SpuWrite\(\)](#), [SpuWrite0\(\)](#), [SpuRead\(\)](#), [SpuSetTransferMode\(\)](#), [SpuGetTransferMode\(\)](#), [SpuSetTransferStartAddr\(\)](#), [SpuGetTransferStartAddr\(\)](#), [SpulsTransferCompleted\(\)](#), [SpuSetTransferCallback\(\)](#)

Chapter 16: Serial Input/Output Library

Table of Contents

Functions

AddSIO	16-3
DelSIO	16-4
Sio1Callback	16-5
_sio_control	16-6

AddSIO

Initialize SIO driver.

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	3.6	12/14/98

Syntax

```
long AddSIO(  
int baud)           Communication speed (bps)
```

Explanation

Initializes the SIO driver at the communication speed *baud*.

Return value

1.

See also

[DelSIO\(\)](#)

DelSIO

Delete SIO driver from kernel.

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	3.6	12/14/98

Syntax

long DelSIO(*void*)

Explanation

Deletes the SIO driver from the kernel.

Return value

1.

See also

[AddSIO\(\)](#)

Sio1Callback

Set SIO interrupt callback function.

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	4.0	12/14/98

Syntax

```
int Sio1Callback(
void(*func)())          Callback function
```

Explanation

Defines *func* as the callback to be triggered when an interrupt has been generated by the interrupt factors (CR_DSRIEN, CR_RXIEN, CR_TXIEN) set by `_sio_control (1,1, param)`. If *func* is 0, a callback is not generated.

When an interrupt is generated, the interrupt flag must be cleared using `_sio_control (2,1,0)` or `_sio_control (1,1,CR_ERRRST)`. The next SIO interrupt isn't generated unless the interrupt flag is cleared.

Return value

Address of previously installed callback function.

_sio_control

Issue SIO command

Library	Header File	Introduced	Documentation Date
<i>libsio.lib</i>	<i>libsio.h</i>	3.6	12/14/98

Syntax

```

long _sio_control(
unsigned long cmd           Command
unsigned long arg          Subcommand
unsigned long param)       Argument

```

Explanation

SIO driver control and information acquisition.

Used in detailed communication with the PC and also when the user wishes to suppress debugging data based on the standard output from the library, etc.

Table 16-1: Command Summary

cmd	arg	Function
0	0	Returns driver status (see Table 16-2)
	1	Returns control line status (see Table 16-3)
	2	Returns communications mode (see Table 16-1)
	3	Returns communications speed (bps units)
	4	Reads 1 byte
1	0	System reservation
	1	Sets param value as control line status (see Table 16-3)
	2	Sets param value as communications mode (see Table 16-4)
	3	Sets param value as communications speed (bps units)
	4	Writes 1 byte
2	0	Resets driver
	1	Clears driver status error-related bits

Table 16-2: Driver Status

bit		Contents
31-10		Undecided
9	SR_IRQ	1: interrupt on
8	SR_CTS	1: CTS is on
7	SR_DSR	1: DSR is on
6		Undecided
5	SR_FE	1: frame error occurs
4	SR_OE	1: overrun error occurs
3	SR_PERROR	1: parity error occurs
2	SR_TXU	1: no communications data
1	SR_RXRDY	1: able to read communications data
0	SR_TXRDY	1: able to write communications data

Table 16-3: Control Line Status

bit	Contents
31-2	undecided
1	1: RTS is on
0	1: DTR is on

Table 16-4: Communications Mode

bit		Contents
31-8		Undecided
7,6		stop bit length
	MR_SB_01	01: 1
	MR_SB_10	10: 1.5
	MR_SB_11	11: 2
5	MR_P_EVEN	parity 2 (1: odd 0: even)
4	MR_PEN	parity 1 (1: exists)
3,2		character length
	MR_CHLEN_5	00: 5 bit
	MR_CHLEN_6	01: 6
	MR_CHLEN_7	10: 7
	MR_CHLEN_8	11: 8
1		Always 1
0		Always 0

Return value

Described in Table 16-1 above.

Table 16-5: Control Register

bit		Contents
31-13		Undecided
12	CR_DSRIEN	1: DSR Interrupt Permission
11	CR_RXIEN	1: Receive Interrupt Permission
10	CR_TXIEN	1: Transmission Interrupt Permission
9,8		0 Fixed
7		Undecided
6	CR_INTRST	1: SIO1 Reset
5	CR_RTS	1: RTS is on
4	CR_ERRRST	1: Interrupt and error flag clear
3		Undecided
2	CR_RXEN	1: Receive permission
1	CR_DTR	1: DTR is on
0	CR_TXEN	1: Transmission permission

Chapter 17: HMD Library

Table of Contents

Structures

GsARGUNIT	17-3
GsARGUNIT_ANIM	17-4
GsARGUNIT_GND...	17-5
GsARGUNIT_IMAGE	17-6
GsARGUNIT_JntMIMe	17-7
GsARGUNIT_NORMAL	17-8
GsARGUNIT_RstJntMIMe	17-9
GsARGUNIT_RstVNMIMe	17-10
GsARGUNIT_SHARED	17-11
GsARGUNIT_VNMIMe	17-12
GsCOORDUNIT	17-13
GsRVIEWUNIT	17-14
GsSEH	17-15
GsSEQ	17-16
GsTYPEUNIT	17-18
GsUNIT	17-20
GsVIEWUNIT	17-21
GsWORKUNIT	17-22

Functions

GsGetHeadpUnit	17-23
GsGetLsUnit	17-24
GsGetLwsUnit	17-25
GsGetLwUnit	17-26
GsInitRstNrmMIMe	17-27
GsInitRstVtxMIMe	17-28
GsLinkAnim	17-29
GsMapCoordUnit	17-30
GsMapUnit	17-31
GsScanAnim	17-32
GsScanUnit	17-33
GsSetRefViewLUnit	17-34
GsSetRefViewUnit	17-35
GsSetViewUnit	17-36
GsSortUnit	17-37
GsU_...	17-38
GsU_03000000	17-39
GsU_03000001...	17-41
GsU_03010110...	17-43
GsU_040100...	17-45

Structures

GsARGUNIT

Primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
} GsARGUNIT;
```

Explanation

The common arguments passed to a primitive driver called from GsSortUnit().

For high speed operation, use the scratch pad.

See also

[GsSortUnit\(\)](#), [GsU_...\(\)](#), [GsU_040100...\(\)](#)

GsARGUNIT_ANIM

Animation driver argument area.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

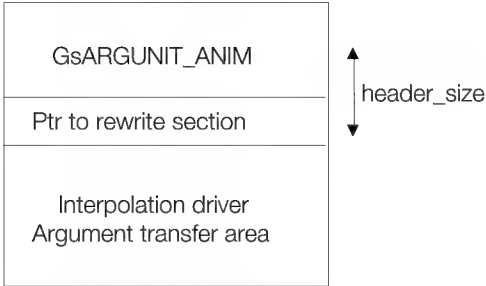
```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    long header_size;        Size of primitive header used in animation
    u_long *htop;            Start address of interpolation function table section
    u_long *ctop;            Start address of sequence control section
    u_long *ptop;            Start address of parameter section
} GsARGUNIT_ANIM;
```

Explanation

The arguments passed to an animation primitive driver called from GsSortUnit(). In addition to the common arguments in GsARGUNIT, it also contains the start address for each section needed for the primitive header size and animation.

The argument transfer area is larger than this structure. A pointer to the section where rewriting is carried out follows GsARGUNIT_ANIM. This pointer plus htop, ctop, and ptop are included in header_size.

The transfer area for the interpolation function follows the pointer to the rewriting section.



The size and meaning of the interpolation driver transfer area depends on the type of interpolation function. the following four parameters apply to linear interpolation:

- 1. Sequence pointer start address
- 2. Interpolation source parameter address.
- 3. Interpolation destination parameter address.
- 4. Address which remembers parameters after interpolation.

For high speed operation, use the scratch pad.

See also

GsSortUnit(), GsU_03000001...(), GsU_03000000(),GsU_03010110...()

GsARGUNIT_GND...

HMD ground primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.1	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when sorting OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_long *polytop;         Pointer to start of ground POLYGON section
    u_long *boxtop;          Pointer to start of ground box section
    u_long *pointtop;        Pointer to start of ground vertex section
    SVECTOR *nortop;         Pointer to start of NORMAL section
} GsARGUNIT_GND;
```

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when sorting OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_long *polytop;         Pointer to start of ground POLYGON section
    u_long *boxtop;          Pointer to start of ground box section
    u_long *pointtop;        Pointer to start of ground vertex section
    SVECTOR *nortop;         Pointer to start of NORMAL section
    u_long *uvtop;           Pointer to start of UV section
} GsARGUNIT_GNDT;
```

Explanation

The arguments passed to a ground primitive driver called from GsSortUnit(). GsARGUNIT_GNDT uses texture, while GsARGUNIT_GND does not.

This structure includes pointers to the start of the ground POLYGON section, the box section, and the vertex section within HMD data, in addition to the parameters in [GsARGUNIT](#).

Use the scratch pad to improve performance.

See also

[GsSortUnit\(\)](#), [GsU_...\(\)](#)

GsARGUNIT_IMAGE

Image primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_long *imagetop;        Pointer to pixel data start
    u_long *cluttop;         Pointer to clut data start
} GsARGUNIT_IMAGE;
```

Explanation

The arguments passed to an image primitive drive called from GsSortUnit(). In addition to the GsARGUNIT members, it includes the pointer to the start of the clut data and texture image pixel data.

For high speed operation, use the scratch pad.

See also

GsSortUnit(),GsU_...()

GsARGUNIT_JntMIMe

Joint MIMe primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_long *coord_sect;      Pointer to COORDINATE section top
    long *mimepr;            Pointer to mime coefficient area
    u_long mimenum;          Number of mime keys
    u_short mimeid;          MIMeID
    u_short reserved;
    u_long *mime_diff_sect;  Pointer to MIMe DIFF section
} GsARGUNIT_JntMIMe;
```

Explanation

The arguments passed to a joint MIMe primitive driver called from GsSortUnit(). In addition to the [GsARGUNIT](#) parameters, it contains a pointer to the coordinate section start within the HMD data, the pointer to the mime coefficient area, the number of mime keys, the MIMeID, and the MIMe DIFF section pointer.

For high-speed operation, use the scratch pad.

See also

[GsSortUnit\(\)](#), [GsU_...\(\)](#), [GsU_040100...\(\)](#)

GsARGUNIT_NORMAL

Independent primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_long *primtop;          Pointer to POLYGON section top
    SVECTOR *vertop;         Pointer to VERTEX section top
    SVECTOR *nortop;         Pointer to NORMAL section top
} GsARGUNIT_NORMAL;
```

Explanation

The arguments passed to an independent primitive driver called from GsSortUnit(). In addition to the GsARGUNIT parameters, it contains the POLYGON section and VERTEX sections within the HMD data, and the pointer to the NORMAL section start.

For high-speed operation, use the scratch pad.

See also

GsSortUnit()

GsARGUNIT_RstJntMIMe

Joint mime reset primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_long *coord_sect;      Pointer to COORDINATE section top
    u_short mimeid;          MIMeID
    u_short reserved;
    u_long *mime_diff_sect;  Pointer to MIMe DIFF section
}
```

GsARGUNIT_RstJntMIMe;

Explanation

The arguments passed to a joint MIMe reset primitive driver called from GsSortUnit(). In addition to the GsARGUNIT parameters, it contains the pointer to the start of the COORDINATE section within the HMD data, the MIMeID and the MIMe DIFF section pointer.

For high-speed operation, use the scratch pad.

See also

GsSortUnit(), GsU_040100...()

GsARGUNIT_RstVNMIMe

Argument area of vertex normal MIMe reset primitive driver.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    u_short mimeid;          MIMeID
    u_short reserved;
    u_long *mime_diff_sect;   MIMe DIFF section pointer
    SVECTOR *orgs_vn_sect;    OrgsVN section pointer
    SVECTOR *vert_sect;       Vertex section pointer
    SVECTOR *norm_sect;       Normal section pointer
} GsARGUNIT_RstVNMIMe;
```

Explanation

The arguments passed to a vertex normal primitive driver called from GsSortUnit. In addition to the GsARGUNIT parameters, it contains the MIMeID within the HMD data, the MIMe DIFF section pointer, the OrgsVN section pointer, the vertex section pointer, and the normal section pointer.

For high-speed operation, use the scratch pad.

See also

GsSortUnit(),GsU_...(), GsU_040100...()

GsARGUNIT_SHARED

Shared primitive driver argument area.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    u_long *primtop;         Pointer to unused packet area
    SVECTOR *vertop;         Pointer to POLYGON section top
    GsWORKUNIT *vertop2;     Pointer to shared VERTEX section top
    SVECTOR *nortop;         Pointer to area storing the calculation results of shared vertex section
    SVECTOR *nortop2;        Pointer to shared NORMAL section top
} GsARGUNIT_SHARED;        Pointer to area storing the calculation results of the shared NORMAL
                             section
```

Explanation

The arguments passed to a shared primitive driver called from GsSortUnit(). In addition to the [GsARGUNIT](#) parameters, it contains the POLYGON section and VERTEX section within the HMD data, the pointer to the NORMAL section start, and the pointer to the area storing those calculation results.

For high-speed operation, use the scratch pad.

See also

[GsSortUnit\(\)](#), [GsU_...\(\)](#)

GsARGUNIT_VNMIMe

Vertex normal mime primitive driver argument area.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    u_long *primp;           Primitive start address
    GsOT *tagp;              Pointer to current GsOT
    int shift;               Number of bits to shift when assigning OT
    int offset;              OT screen coordinate system Z-axis offset
    PACKET *out_packetp;     Pointer to unused packet area
    long *mimepr;            Pointer to mime coefficient area
    u_long mimenum;          Number of mime keys
    u_short mimeid;          MIMeID
    u_short reserved;
    u_long *mime_diff_sect;  MIMe DIFF section pointer
    SVECTOR *orgs_vn_sect;   OrgsVN section pointer
    SVECTOR *vert_sect;      Vertex section pointer
    SVECTOR *norm_sect;      Normal section pointer
} GsARGUNIT_VNMIMe;
```

Explanation

The arguments passed to the vertex normal mime primitive driver called from GsSortUnit(). In addition to the GsARGUNIT parameters, it contains the MIMeID within the HMD data, the MIMe DIFF section pointer, the OrgsVN section pointer, the vertex section pointer, and the normal section pointer.

For high-speed operation, use the scratch pad.

See also

GsSortUnit(),GsU_040100...()

GsCOORDUNIT

Matrix type coordinate system.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	9/1/99

Structure

```
struct _GsCOORDUNIT {
    u_long flag;                Flag indicating whether matrix was rewritten
    MATRIX matrix;              Matrix
    MATRIX workm;               Result of multiplication from this coordinate system to the
                                WORLD coordinate system
    SVECTOR rot;               Rotation vector for creating matrix
    struct _GsCOORDUNIT *super; Pointer to parent coordinates
} GsCOORDUNIT;
```

Explanation

GsCOORDUNIT has parent coordinates and is defined by *matrix*. *workm* retains the result of multiplication of matrices performed by GsGetLwUnit() and GsGetLsUnit() in each node of GsCOORDINATE2 using the WORLD coordinates. However, this result is not stored in the *workm* of the coordinate system directly connected to the WORLD coordinate system.

GsGetLwUnit() and GsGetLsUnit() use *flag* to omit calculations for a node when they have already been performed. The external variable PSDCNT sets the flag and 0 clears it. If you change the contents of *matrix*, you must clear this flag. If it is not cleared, GsGetLwUnit() and GsGetLsUnit() will fail to execute normally.

See also

[GsGetLwUnit\(\)](#), [GsGetLsUnit\(\)](#), [GsGetLwsUnit\(\)](#), [GsMapCoordUnit\(\)](#)

GsRVIEWUNIT

HMD viewpoint position (Reference type).

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    long vpx, vpy, vpz;           Viewpoint coordinates
    long vrx, vry, vrz;           Reference point coordinates
    long rz;                       Viewpoint twist
    GsCOORDUNIT *super;           Pointer to coordinate system which sets viewpoint (GsCOORDUNIT
                                type)
} GsRVIEWUNIT;
```

Explanation

GsRVIEWUNIT contains viewpoint information and is set by GsSetRefViewUnit(). The viewpoint coordinates in the coordinate system displayed by *super* are set in *vpx*, *vpy* and *vpz*. The reference point coordinates in the coordinate system displayed by *super* are set in *vrx*, *vry* and *vrz*.

When the z axis is a vector from the viewpoint to the reference point, *rz* specifies the screen inclination against the z axis in fixed decimal format, with 4096 set to one degree. Viewpoint and reference point coordinate systems are set in *super*. For an example of the use of this function, an airplane cockpit view can be realized simply by setting *super* to the airplane coordinate system.

See also

[GsSetRefViewUnit\(\)](#), [GsSetRefViewLUnit\(\)](#)

GsSEH

HMD animation sequence header.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    short idx;           Indexes the sequence control descriptor of the sequence header and
                        represents it in the index within the sequence control section
    u_char sid;          Sequence ID
    u_char pad;          System reservation
} GsSEH;
```

Explanation

Contains sequence information. Multiple sequences are stored as an array of GsSEH after the sequence pointer.

idx provides the index to the sequence control descriptor of the sequence playback start. This index is set to GsSEQ.*ti*. By setting GsSEH.*sid* to GsSEQ.*sid*, playback of the sequence can be started.

When multiple sequences are present, select one from the GsSEH array and update the value to [GsSEQ](#).

pad may be freely used if TOD animation is not used.

See also

[GsLinkAnim\(\)](#), [GsU_03000000\(\)](#), [GsU_03000001...\(\)](#)

GsSEQ

HMD animation sequence pointer.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_long rewrite_idx;
    u_short size;
    u_short num;
    u_short ii;
    u_short aframe;
    u_char sid;
    char speed;
    u_short srcii;
    short rframe;
    u_short tframe;
    u_short ci;
    u_short ti;
    u_short start;
    u_char start_sid;
    u_char pad;
} GsSEQ;
```

The upper 8 bits are a word offset into the animation primitive header, which contains a pointer the section to be updated. The lower 24 bits are the relative offset (in word units) within that section.

Size up to area where next sequence pointer is stored.

Number of sequences stored in this sequence pointer.

Index of area which stores parameters after interpolation (Relative offset within parameter section)

Frame count of sequence. Automatically decreased by the frame update driver. The sequence stops at 0. For endless playback, set it to 0xffff, and the decrease is prohibited.

Current status sequence ID of the currently playing sequencer. Matching can be achieved when performing sequence jump and END operations with the sequence ID value set from 0 to 127. The sequence flow can be controlled using SID. When performing a jump description with the sequence control descriptor, it defines the match SID and the SID of the location to be jumped to. By setting the match SID to 0, it is possible to match all SIDs.

Controls sequence playback speed. The two's complement of the sign bit indicates the direction of playback. Standard playback speed is 0x10. 0x7f increases speed eight times, and 0x01 makes speed 1/16. The sign bit is the bit used for reverse playback. The packaging is carried out by subtracting *speed* from *rframe* frame by frame to make a new RFRAME value.

Retains data which should be specified in *ii*. The area for parameter saving already included in the data is retained by the index.

In the interpolation coefficient 0 between the key frames SRC FRAME, *rframe*==*tframe* becomes DST FRAME. It is a 12 bit fixed-point number. The distance between key frames, as a 12-bit fixed-point number. The *tframe* in the sequence control descriptor is 8 bits and shifts four bits to the left when updated to *GsSEQ.tframe*. When *tframe* is 0, the unobtained frame update driver passes the next key frame in DST KEYFRAME (the interpolated target key frame).

Index to the parameter which retains source key frames (Word offset within parameter section)

Parameter index which retains destination keyframes (Word offset within parameter section)

The sequence start index. *start* is copied to *ti* and *rframe* set to zero when sequence begins. The work area indicated for Bspline and related interpolation is set in this member.

Stream ID of sequence to be started. *start_sid* is copied to *sid* when sequence begins.

Clears to 0 when the key frame is switched. By setting it to a non-zero value, the program can detect when interpolation has completed.

Explanation

GsSEQ contains the frame update driver internal status. By rewriting this structure during execution, animation can be dynamically controlled.

GsSEQ structures in the HMD data can be indicated using *GsLinkAnim()*. The following members are only referred to by the frame update driver: *rewrite_idx*, *size*, *num*, *ii*, *speed*, *srcii*, *start* and *start_sid*. All others are rewritten by the frame update driver.

See also

[GsU_03000000\(\)](#), [GsU_03000001...\(\)](#), [GsLinkAnim\(\)](#)

GsTYPEUNIT

Type storage area.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Structure

```
typedef struct {
    u_long type;           Primitive type
    u_long *ptr;           Address in HMD data where type is stored
} GsTYPEUNIT;
```

Explanation

This structure is passed to `GsScanUnit()`. By assigning the subordinate functions corresponding to the type to `ptr`, the type is overwritten on the subordinate functions pointer. `GsSortUnit()` calls that subordinate function.

The structure of *type* is as follows:

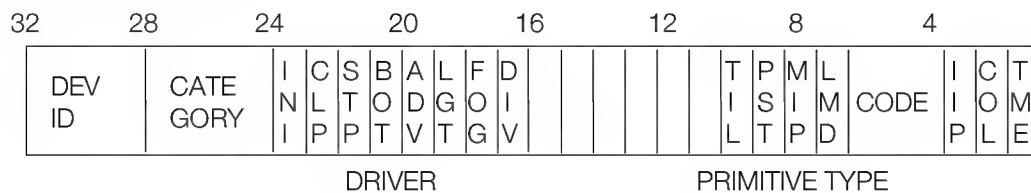


Table 17-1

DEV ID:	0000: SCE Reserved
CATEGORY:	0000: Polygon 0001: Shared Polygon 0010: Image data 0011: Animation 0100: MIMe 0101: Ground 0111: Equipoment
DRIVER:	
INI(init)	0: None 1: COORDINATE initialization is needed
CLP(clip)	0: No clipping 1: No clipping on left and top of screen
STP(Semi-trans)	0: Semi-transparent 1: Semi-transparent
BOT(both-side)	0: One-sided polygon 1: Two-sided polygon
ADV(active-div)	0: No automatic division 1: With automatic division
LGT(light)	0: With light source calculation 1: Without light source calculation
FOG(fog)	0: FOG off 1: FOG on
DIV(divide)	0: Without division 1: With division

PRIMITIVE TYPE:

TILE	0: No information for continous texture 1: With information for continous texture
PST(preset)	0: No presets 1: With presets
MIP(mip-map)	0: No mip-map 1: With mip-map
LMD(light-mode)	0: With normals 1: No normals
CODE	000: Reserved 001: Triangle 010: Quadrangle 011: strip mesh 100-111: Reserved
IIP	0: Flat 1: Gouraud
COL(colored)	0: 1 quality color within identical polygon 1: Color for every vertex
TME	0: Texture mapping OFF 1: Texture mapping ON

See also

[GsMapCoordUnit\(\)](#), [GsScanUnit\(\)](#), [GsSortUnit\(\)](#)

GsUNIT

Three-dimensional object handler for GsSortUnit().

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    GsCOORDUNIT *coord;      Pointer to local coordinate system
    u_long *primtop;         Pointer to primitive block header
} GsUNIT;
```

Explanation

GsUNIT exists in every HMD data primitive block and allows movement of three-dimensional models. GsSortUnit() is used to register GsUNIT to the ordering table. *coord* is the pointer to the primitive block individual coordinate system. By setting the matrix in the coordinate system pointed to by *coord*, the object's location, inclination and size are reflected.

The primitive block start address is passed in *primtop*.

See also

GsSortUnit(), GsU_00...()

GsVIEWUNIT

HMD viewpoint position (matrix type).

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>12/14/98</i>

Structure

```
struct GsVIEWUNIT {
    MATRIX view;           Matrix used to change from parent coordinates to viewpoint coordinates
    GsCOORDUNIT *super;   Pointer to coordinate system which sets viewpoint
};
```

Explanation

This structure sets the viewpoint coordinates used by HMD. It directly specifies the matrix used to change from parent coordinates to viewpoint coordinates. The function used to set GsVIEWUNIT is GsSetViewUnit().

See also

[GsSetViewUnit\(\)](#)

GsWORKUNIT

Calculation result storage area for shared primitive.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Structure

```
typedef struct {
    DVECTOR vec;           Area which stores the thre-dimensional coordinate values and the two-
                           dimensional coordinate values after perspective conversion.
    short otz;             Area which stores the OTZ value obtained when vec is requested
    short p;               Area which stores the interpolation value obtained when vec is requested
} GsWORKUNIT;
```

Explanation

When using a shared primitive, first a perspective conversion of each three-dimensional coordinate value is carried out by its matrix to convert it into a two-dimensional coordinate. After perspective conversion is completed by the matrices, the shared primitive refers to the converted two-dimensional coordinate and creates a packet. This structure is the area which stores the results of that calculation.

See also

Functions

GsGetHeadpUnit

Obtain current HMD header address.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

u_long *GsGetHeadpUnit(void)

Explanation

Returns the current type header address when using GsScanUnit() to scan HMD data.

Return value

The current type header address.

See also

[GsScanUnit\(\)](#)

GsGetLsUnit

Calculate a local screen matrix.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>9/1/99</i>

Syntax

```
void GsGetLsUnit(
  GsCOORDUNIT *coord,    Local coordinates
  MATRIX *m)              Matrix
```

Explanation

Calculates a local screen perspective transformation matrix from the GsCOORDUNIT structure pointed to by *coord* and stores the result in the MATRIX structure pointed to by *m*.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When GsGetLsUnit() is called next, calculations up to the node to which no changes have been made are omitted. This is controlled by a GsCOORDUNIT member flag (libgs replaces the external variable PSDCNT in flags already calculated by GsCOORDUNIT).

If the contents of a superior node are changed, the effect on a subordinate node is handled by libgs, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

GsGetLwsUnit

Calculate local world and local screen matrices.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

```
void GsGetLwsUnit(
  GsCOORDUNIT *coord,   Pointer to local coordinates
  MATRIX *lw,           Matrix which stores local world coordinates as the result
  MATRIX *ls)           Matrix which stores local screen coordinates as the result
```

Explanation

Calculates both local world coordinates and local screen coordinates. It is faster than calling GsGetLwUnit() and then calling GsGetLsUnit(). This is a quick way of obtaining a local world matrix (*lw*) to pass to GsSetLightMatrix() to carry out light source calculations.

See also

[GsGetLwUnit\(\)](#), [GsGetLsUnit\(\)](#)

GsGetLwUnit

Calculate local world matrix.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>9/1/99</i>

Syntax

```
void GsGetLwUnit(
  GsCOORDUNIT *coord,   Local coordinates
  MATRIX *m)            Matrix
```

Explanation

Calculates a local screen perspective transformation matrix from the GsCOORDUNIT structure pointed to by *coord* and stores the result in the MATRIX structure pointed to by *m*.

For high speed operation, the function retains the result of calculation at each node of the hierarchical coordinate system. When GsGetLwUnit() is called next, calculations up to the node to which no changes have been made are omitted. This is controlled by a GsCOORDUNIT member flag (libgs replaces the external variable PSDCNT in flags already calculated by GsCOORDUNIT).

If the contents of a superior node are changed, the effect on a subordinate node is handled by libgs, so it is not necessary to clear the flags of all subordinate nodes of the changed superior node.

See also

[GsGetLsUnit\(\)](#), [GsGetLwsUnit\(\)](#)

GsInitRstNrmMIMe

Initialize HMD normal MIMe driver.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.1	12/14/98

Syntax

```
void GsInitRstNrmMIMe(  
u_long *prmtop,           Primitive start address  
u_long *hp)               Primitive header start address
```

Explanation

Initializes the HMD library normal MIMe driver.

When using the normal MIMe reset driver (GsU_04010029), it is necessary to initialize data using this function. When the address of the primitive driver is placed in the *ptr* of the GsTYPE structure which is returned by GsScanUnit(), GsInitRstNrmMIMe() is called.

Refer to: psx\sample\graphics\hmd\mime\.

See also

[GsInitRstVtxMIMe\(\)](#), [GsU_04010010....\(\)](#)

GsInitRstVtxMIMe

Initialize HMD vertex MIMe driver.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.1	12/14/98

Syntax

void GsInitRstVtxMIMe(

u_long *primtop, Primitive start address

u_long *hp) Primitive header start address

Explanation

Initializes the HMD library vertex MIMe driver.

When using the vertex MIMe reset driver (GsU_04010028), it is necessary to initialize data using this function. When the address of the primitive driver is placed in the *ptr* of the GsTYPEUNIT structure which is returned by GsScanUnit(), GsInitRstVtxMIMe() is called..

Refer to: psx\sample\graphics\hmd\mime\.

See also

[GsInitRstNrmMIMe\(\)](#), [GsU_04010010....\(\)](#)

GsLinkAnim

Link GsSEQ array and HMD data.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>8/9/99</i>

Syntax

```
int GsLinkAnim(  
GsSEQ *seq,           GsSEQ structure array address  
u_long *p)            Address of first type animation section
```

Explanation

Links a GsSEQ structure, which contains information to control one sequence, to corresponding data in an HMD file. As with GsScanAnim(), GsLinkAnim() must be activated during global SCAN.

By using *seq*, the programmer can control animation playback in real time. With GsLinkAnim(), specific members of specific sequences such as *seq[3]->aframe = 100* can be operated after switching to GsLinkAnim().

Information on each individual sequence can be accessed in the sequence header GsSEH. For example, assuming that ten sequences are defined in the fifth sequence, in order to play back the fourth, the fourth start index can be transmitted in the sequence pointer as follows:

```
seq[5]->ti = ((GsSEQ *)&seq[5].start)+3->sid;
```

Return value

Returns the linked number.

See also

[GsScanAnim\(\)](#)

GsMapCoordUnit

Map COORDINATE within the HMD data to actual address.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>12/14/98</i>

Syntax

GsCOORDUNIT *GsMapCoordUnit(
 u_long *base, HMD data start address
 u_long *p) Type of address where INI (init) bits are set up

Explanation

In cases where COORDINATE exists within the HMD data, one type in which INI bits are set up exists in **GsTYPEUNIT** type when GsScanUnit() is carried out. In such cases, by transferring that type address to GsMapCoordUnit(), the COORDINATE TOP within the data and super within COORDINATE are converted to the actual address.

Return value

COORDINATE section start address.

See also

[GsMapUnit\(\)](#), [GsScanUnit\(\)](#)

GsMapUnit

Map HMD data offset to actual address.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

```
void GsMapUnit(
u_long *p)          HMD data start address
```

Explanation

In HMD data, sections are referred to by pointers. When creating HMD data, the pointers are specified as word offsets from the start of HMD data, because the exact memory locations are not known. GsMapUnit() converts these into actual addresses, so the HMD data can be used.

A flag in the HMD header records whether addresses have been mapped, so there are no adverse effects even if GsMapUnit() is called again.

See also

[GsScanUnit\(\)](#), [GsSortUnit\(\)](#)

GsScanAnim

Perform SCAN for HMD animation.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Syntax

u_long *GsScanAnim(
u_long *p, Address where animation section first type exists.
 When *p* is 0, the type following that previously called by GsScanAnim() is
 examined
GsTYPEUNIT *ut) Pointer to area in which the type and its address are stored

Explanation

HMD data scanning is divided into two parts:

- GsScanUnit() carries out a common scan of all sections (a global scan).
- A dedicated scan function carries out a local scan of each section. GsScanAnim() is a dedicated scan function for HMD animation.

Scanning should be carried out after the animation type has been globally scanned. Since a bit is attached to the INI field of the animation type which was first globally scanned at the authoring level, timing for performing a local scan can be gauged by closely watching that bit.

GsScanAnim() is called to scan the types of all animation sections using the following procedure:

1. If the Type selected in the global SCAN is category 3 (animation) and the INI bit is 1, a local SCAN should be performed (Type=0x03800000).
2. GsScanAnim() issues the above-mentioned type address as *p*.
3. If the return value is 0, data is wrong. If it is 1, issue GsScanAnim() once more.
4. An interpolation primitive driver function pointer corresponding to the selected *ut.type* is replaced by *ut.ptr*.
5. Return to #3 and repeat until 0 is returned.

With the primitive driver pointer which was written over by the HMD data, there are no adverse effects even if GsScanAnim() is called again.

Return value

- 0 type to be scanned does not exist
- 1 Scan successful, type exists

See also

GsScanUnit(), GsLinkAnim()

GsScanUnit

Examine types within HMD data.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

```
int GsScanUnit(
u_long *p,           Block start address. When it is 0, the type following that previously called by
                     GsScanUnit() is examined
GsTYPEUNIT *ut,      Pointer to area where the type and its address are stored
GsOT *ot,             Pointer to the OT
u_long *scratch)      Specifies scratch pad address
```

Explanation

HMD data is divided by type and primitive drivers are called according to type. By overwriting the type by address of the primitive driver, the primitive driver can be called during GsSortUnit(). On examination of the contents of GsTYPEUNIT returned by GsScanUnit() the pointer to the primitive driver is replaced by one which can be freely used by the user.

GsScanUnit() internally copies the type corresponding to the header to *scratch*. By passing *scratch* as a primitive driver argument, the primitive driver can be activated. Generally, image primitives which are only activated once (such as GsUIMGO and GsUIMG1) are activated at this initialization.

Initially, set *p* to the block start address when calling GsScanUnit(); subsequent calls should set *p* to 0, and it will scan to the end of the block. This operation is carried out for every block.

Since a flag is attached to the primitive driver pointer which was written over by the HMD data, there are no adverse effects even if GsScanUnit() is called again.

Return value

- 0 Reached the end of the block, type does not exist
- 1 Type exists

GsSetRefViewLUnit

Set HMD viewpoint position (High precision).

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>12/14/98</i>

Syntax

```
int GsSetRefViewLUnit(
  GsRVIEWUNIT *pv)    Viewpoint position information (viewpoint observation point type)
```

Explanation

Calculates GsWSMATRIX from viewpoint information. Its parameter is the GsRVIEWUNIT structure. If the viewpoint is not moved, GsWSMATRIX doesn't change, so in such cases there is no need to call each frame.

However, if the viewpoint is moved, the changes aren't reflected unless each frame is called. Although the number of calculation mistakes using this function is smaller than with GsSetRefViewUnit(), the execution time is double. When the GsRVIEWUNIT *super* member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint moves if the parent coordinate system parameters are changed. In such cases, GsSetRefViewLUnit() must be called for each frame.

Return value

If viewpoint setting is successful:0; if unsuccessful:1.

GsSetRefViewUnit

Set HMD viewpoint position.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

```
int GsSetRefViewUnit(
  GsRVIEWUNIT *pv)      Viewpoint position information (viewpoint observation point type)
```

Explanation

Calculates GsWSMATRIX from viewpoint information. If the viewpoint isn't moved, GsWSMATRIX doesn't change, so in such cases there is no need to call each frame.

However, if the viewpoint is moved, the changes aren't reflected unless each frame is called. When the GsRVIEWUNIT *super* member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint moves if the parent coordinate system parameters are changed. In such cases, GsSetRefViewUnit() must be called for each frame..

Return value

If viewpoint setting is successful:0; if unsuccessful:1.

GsSetViewUnit

Set HMD viewpoint.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	<i>4.0</i>	<i>12/14/98</i>

Syntax

```
int GsSetViewUnit(
  GsVIEWUNIT *pv)      Viewpoint position information (matrix type)
```

Explanation

Directly sets GsWSMATRIX. If you use GsSetRefViewUnit() to determine GsWSMATRIX from the viewpoint and the focal point, insufficient precision may cause errors when you move the viewpoint; it is more effective to use GsSetViewUnit(). When the GsVIEWUNIT *super* member is set to anything besides WORLD, even if the other parameters remain unchanged, the viewpoint moves if the parent coordinate system parameters are changed. In such cases, you must call GsSetViewUnit() for each frame. If GsIDMATRIX2 is used as the base matrix, then the aspect ratio of the screen is adjusted automatically.

Return value

0, if setting is successful; 1, if unsuccessful.

GsSortUnit

Allocate an object to the ordering table.

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Syntax

```
void GsSortUnit(  
  GsUNIT *objp,           Pointer to an object  
  GsOT *otp,              Pointer to the OT  
  u_short *scratch)       Specifies scratch pad address
```

Explanation

Performs perspective transformation and light source calculation on a three dimensional object handled by GsUNIT. The rendering command is generated in the packet area specified by GsSetWorkBase(). The rendering command generated is then Z-sorted and allocated to the OT displayed by otp. When scratch calls the subordinate functions, it is used in the transfer of variables. Usage volume is as follows:

Table 17-2

Type	Scratch area usage (Unit: Byte)
Independent polygons	32
Shared polygons	40
Fixed division independent triangles	568
Fixed division independent quadrangles	852

GsU_...

GsSortUnit() primitive driver group (Polygon, Shared Polygon, Image, Ground).

Library	Header File	Introduced	Documentation Date
libhmd.lib	libhmd.h	4.0	12/14/98

Syntax

- u_long *GsU_00.....(
GsARGUNIT *ap)

Polygon primitive driver group
Start address for arguments transfer area
- u_long *GsU_01.....(
GsARGUNIT *ap)

Shared polygon primitive driver group
Start address for arguments transfer area
- u_long *GsU_02.....(
GsARGUNIT *ap)

Image primitive driver group
Start address for arguments transfer area
- u_long *GsU_05.....(
GsARGUNIT *ap)

Ground primitive driver group
Start address for arguments transfer area

Explanation

This is the GsSortUnit() primitive driver group.

When initializing HMD data for use, the GsTYPEUNIT structure type returned by GsScanUnit() must be checked, and the primitive driver address must be placed in the pointer. The primitive drivers currently supported are listed in the HMD section in the *File Formats* manual.

When using a primitive driver which performs division, the number of divisions must be set to the significant 8 bits of the area in which the primitive index is stored as follows:

Type	
# of polygons	Size
# of divisions	Primitive index

In order to specify the number of divisions, a macro such as the one following is provided in libhmd.h.

Table 17-3

Macro	#of divisions
GsUNIT_DIV1	2x2 divisions
GsUNIT_DIV2	4x4 divisions
GsUNIT_DIV3	8x8 divisions
GsUNIT_DIV4	16x16 divisions
GsUNIT_DIV5	32x32 divisions

Furthermore, when using automatic division, it is necessary to use GsSetAzwh() and to set the division condition for the z value, polygon size, etc.

Refer to the sample program: (psx\sample\graphics\hmd\pdriver\00000008.c).

Return value

Start address of the next primitive driver.

See also

GsSortUnit(), GsScanUnit()

GsU_03000000

HMD animation frame update driver.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

```
u_long *GsU_03000000(
GsARGUNIT_ANIM *sp)      Argument transfer area
```

Arguments**Explanation**

The frame update driver interprets the sequence descriptor and calls the appropriate interpolation function. It carries out sequence jumps, reverses, etc. Since the driver's internal status is stored in each sequence pointer area, real-time control of the sequence is possible if the programmer rewrites this area when executing the program.

For high speed operation, specify scratch pad. The argument information built on *sp* is as follows:

primtop
tag(OT)
shift(OT)
OUTP(packet area)
Animation header size
Interpolation function header
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
VERTEX TOP pointer

Sections except the last two are always set. Since the primitive headers of the other sections are copied as is, if the header shape changes, it will be altered. The animation header size contains the amount of animation information which is copied to the arguments area. In the case of the above example, 6.

Refer to the explanation in [GsSEQ](#) for details on the sequence pointer.

The frame update driver is linked to the HMD primitive block PRE-PROCESS.

GsSortUnit() is called when processing the first primitive block (PRE-PROCESS). The frame update driver specifies the sequence descriptor which should be referred to by the sequence pointer. By interpreting that sequence pointer the sequence progression is controlled. Finally, the sequence descriptor which refers to the key frame performs interpretation, a suitable interpolation driver (SRC FRAME or DST FRAME) is attached and called.

Refer to the sample program, (PS-X/sample/graphics/hmd/pdriver/GsU_00000008.c).

Return value

The area in which the start address of the next primitive driver is stored.

See also

[GsSortUnit\(\)](#), [GsScanUnit\(\)](#)

GsU_03000001...

HMD animation interpolation function (alignment, COORDINATE).

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

```
int GsU_03000001...(
GsARGUNIT_ANIM *sp)    Argument transfer area
```

Arguments**Explanation**

This function is the interpolation driver for parameter interpolation.

For high-speed operation, use the scratch pad. Information on arguments contained in *sp* is as follows:

primtop
tag(OT)
shift(OT)
offset(OT)
OUTP(packet area)
Animation header size
Interpolation function table pointer
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
VERTEX TOP pointer
Sequence pointer address
Pointer to src parameters
Pointer to dst parameters
Write pointer after interpolation

The parameters in the interpolation function which are always fixed are the last four:

- Sequence pointer address: Designates the section which rewrites data from the rewrite index after interpolation, and the rewrite address.
- Pointer to src parameter: Pointer which indicates the src keyframe within the parameter section.
- Pointer to dst parameter: Pointer which indicates the dst keyframe within the parameter section.
- Pointer which writes after interpolation: There are cases when you wish to retain the parameter value after interpolation for the Realtime Motion Switch. The area retained for that purpose is passed as a pointer which indicates the keyframe within the parameter section. If 0, writing is not performed.

The interpolation coefficient is calculated from the TFRAME and RFRAME stored in the sequence pointer and interpolation is performed.

If src is 0 and dst is 1, 1-RFRAME/TFRAME becomes the interpolation coefficient.

Since the name of the function in the interpolation type indicates the HMD type in interpolation types, refer to the HMD format

The interpolation algorithm can be one of the following:

- **LINEAR:** Interpolate the interval between SRC KEYFRAME and DST KEYFRAME as a straight line.
- **BEZIER:** The KEY FRAME has 3 control points. One control point in the DST KEYFRAME and 3 control points in the SRC KEYFRAME comprise the 4 control points used for BEZIER interpolation.
- **BSPLINE:** The KEY FRAME has one control point. SRC-2, SRC-1, SRC, and DST together comprise 4 control points that are used for BSPLINE interpolation. A WORK area is maintained for the sequence history of SRC-2 and SRC-1. The WORK area is set from the START IDX member of the sequence pointer. 4 x 32 bits are used in the WORK area. At the start of the sequence, there is no history of SRC-2 and SRC-2 so it is necessary to arrange the first 3 key frames so that TFRAME = 0.

The primitive drivers currently supported by libhmd are listed in the HMD section in the *File Formats* manual.

Refer to: psx\sample\graphics\hmd\pdriver\03000001.c, 03000002.c, 03000003.c, 03000010.c, 03000020.c, 03000030.c

Return value

0 after normal interpolation.

If the next key frame is read unconditionally, 1 is returned and in the same frame, the destination key frame is once again called back as an argument. This case allows the sequence to instantly advance with TFRAME=0.

See also

[GsU_03000000\(\)](#)

GsU_03010110...

HMD animation interpolation function (general).

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.1	12/14/98

Syntax

```
int GsU_03010110...(
GsARGUNIT_ANIM *sp)    Argument transfer area
```

Arguments**Explanation**

This function is the generic interpolation driver for parameter interpolation. The numerical values that are interpolated can be packaged as 3-element vectors of 8-bits, 16-bits, or 32-bits, or as a scalar. The argument area pointer is passed as the argument. The interpolation coefficient is calculated from the TFRAME and RFRAME stored in the sequence pointer and interpolation is performed.

For high-speed operation, use the scratch pad. Information on arguments contained in sp is as follows:

prmtop
tag(OT)
shift(OT)
offset(OT)
OUTP(packet area)
Animation header size
Interpolation function table pointer
CONTROL TOP pointer
PARAMETER TOP pointer
COORDINATE TOP pointer
VERTEX TOP pointer
Sequence pointer address
Pointer to src parameters
Pointer to dst parameters
Write pointer after interpolation

The parameters in the interpolation function which are always fixed are the last four.

- Sequence pointer address: Designates the section which rewrites data from the rewrite index after interpolation, and the rewrite address.
- Pointer to src parameter: Pointer which indicates the src keyframe within the parameter section.
- Pointer to dst parameter: Pointer which indicates the dst keyframe within the parameter section.
- Pointer which writes after interpolation: There are cases when you wish to retain the parameter value after interpolation for the Realtime Motion Switch. The area retained for that purpose is passed as a pointer which indicates the keyframe within the parameter section. If 0, writing is not performed.

If src is 0 and dst is 1, 1-RFRAME/TFRAME becomes the interpolation coefficient.

Since the name of the function in the interpolation type indicates the HMD type in interpolation types, refer to the HMD format

The interpolation algorithm can be either LINEAR, BEZIER, or BSPLINE.

- LINEAR: Interpolate the interval between SRC KEYFRAME and DST KEYFRAME as a straight line.
- BEZIER: The KEY FRAME has 3 control points. One control point in the DST KEYFRAME and 3 control points in the SRC KEYFRAME comprise the 4 control points used for BEZIER interpolation.
- BSPLINE: The KEY FRAME has one control point. SRC-2, SRC-1, SRC, and DST together comprise 4 control points that are used for BSPLINE interpolation.

A WORK area is maintained for the sequence history of SRC-2 and SRC-1. The WORK area is set from the START IDX member of the sequence pointer. 4 x 32 bits are used in the WORK area. At the start of the sequence, there is no history of SRC-2 and SRC-2 so it is necessary to arrange the first 3 key frames so that TFRAME = 0.

The primitive drivers currently supported by libhmd are listed in the HMD section in the *File Formats* manual.

Refer to: psx\sample\graphics\hmd\pdriver\03000001.c, 03000002.c, 03000003.c, 03000010.c, 03000020.c, 03000030.c

Return value

0 after normal interpolation.

If the next key frame is read unconditionally, 1 is returned and in the same frame, the destination key frame is once again called back as an argument. This case allows the sequence to instantly advance with TFRAME=0.

See also

[GsU_03000000\(\)](#)

GsU_040100...

HMD MIMe driver.

Library	Header File	Introduced	Documentation Date
<i>libhmd.lib</i>	<i>libhmd.h</i>	4.0	12/14/98

Syntax

u_long *GsU_040100...(
GsARGUNIT *ap)

Start address of argument transfer area. For high-speed operation, use the scratch pad.

Explanation

This function is the driver group for MIMe animation from the HMD library. Although the *ap* argument is GsARGUNIT type, it is handled internally as a different type. MIMe category drivers currently supported by libghmd are as follows:

Table 17-4

Driver name	Type name macro	ap processing type	Explanation
GsU_04010010	GsJntAxesMIMe	GsARGUNIT_JntMIMe	Axis interpolation joint mime
GsU_04010018	GsRstJntAxesMIMe	GsARGUNIT_RstJntMIMe	Axis interpolation joint mime reset
GsU_04010011	GsJntRPYMIMe	GsARGUNIT_JntMIMe	RPY value interpolation joint mime
GsU_04010019	GsRstJntRPYMIMe	GsARGUNIT_RstJntMIMe	RPY value interpolation joint mime reset
GsU_04010020	GsVtxMIMe	GsARGUNIT_VNMIMe	Vertex mime
GsU_04010021	GsNrmMIMe	GsARGUNIT_VNMIMe	Normal mime
GsU_04010028	GsRstVtxMIMe (*)	GsARGUNIT_RstVNMIMe	Vertex mime reset
GsU_04010029	GsRstNrmMIMe (*)	GsARGUNIT_RstVNMIMe	Normal mime reset

(*) Initialization is needed for GsInitRstVtxMIMe, GsInitRstNrmMIMe

Return value

Start address of next primitive driver.

Refer to sample program (psx\sample\grapihcs\hmd\pdriver\00000008.c).

See also

GsARGUNIT_JntMIMe(), GsARGUNIT_RstJntMIMe(), GsARGUNIT_VNMIMe(), GsARGUNIT_RstVNMIMe(), GsInitRstNrmMIMe(), GsInitRstVtxMIMe()

Chapter 18: PDA Library (libmcx)

Table of Contents

Functions

McxAllInfo	18-3
McxCardType	18-5
McxCurrCtrl	18-6
McxExecApl	18-7
McxExecFlag	18-8
McxFlashAcs	18-9
McxGetApl	18-10
McxGetMem	18-11
McxGetSerial	18-12
McxGetTime	18-13
McxGetUIFS	18-14
McxHideTrans	18-15
McxReadDev	18-16
McxSetLED	18-17
McxSetMem	18-18
McxSetTime	18-19
McxSetUIFS	18-20
McxShowTrans	18-21
McxStartCom	18-22
McxStopCom	18-23
McxSync	18-24
McxWriteDev	18-26

McxAllInfo

Get all PDA information (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxAllInfo (
  int port,           Port number (see Table 18-1)
  unsigned char *state) Pointer to buffer for storing the results of reading all PDA information. The
                      buffer size must be 18 bytes.
```

Explanation

Port numbers are as follows:

Table 18-1: Port numbers

	Port 1	Port 2
Direct connect	0x00	0x10
Multitap A	0x00	0x10
Multitap B	0x01	0x11
Multitap C	0x02	0x12
Multitap D	0x03	0x13

With 1 Vsync this function can simultaneously obtain the block number of the header which contains an executing PDA application, the PDA application's flash memory priority write settings, the state of the PDA's current capacity controls, the serial number, and time and date information.

The contents stored in state are as follows:

Table 18-2

Offset	Contents
0	Executing PDA application number (LSB)
1	Executing PDA application number (MSB)
2	Speaker disabled
3	IR communication disabled
4	PDA serial number (LSB)
5	PDA serial number (1)
6	PDA serial number (2)
7	PDA serial number (MSB)
8	Real-time clock (100 years)
9	Real-time clock (year)
10	Real-time clock (month)
11	Real-time clock (day)
12	Real-time clock (day of the week)
13	Real-time clock (hours)
14	Real-time clock (minutes)
15	Real-time clock (seconds)
16	PDA application's flash write priority disabled
17	LED disabled

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has been completed. Use `McxSync()` to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

Note: It is impossible to check for duplicate registrations between an `McxXXX()` function and functions like `_card_xxx()` and `MemCardxxx()`. When a request is made to register an `McxXXX()` function after these other processes have already been registered, the return value will still be "Registration accepted", even though registration will not be guaranteed. Consequently, it is best to avoid simultaneous registration between an `McxXXX()` function and `_card_xxx()` or `MemCardxxx()` functions. Likewise, if registration of an `McxXXX()` function is requested while executing `read()`, `write()`, `delete()`, etc., after a Memory Card file has been opened, the return value will still be "Registration accepted", even though registration will not be guaranteed. Again, it is best to avoid these cases.

See also

[McxGetApl\(\)](#), [McxCurrCtrl\(\)](#), [McxFlashAcs\(\)](#), [McxGetSerial\(\)](#), [McxGetTime\(\)](#), [McxSync\(\)](#)

McxCardType

Probe PDA connection status (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxCardType (
int port)          Port number (see Table 18-1)
```

Explanation

This function is used to check for the presence of a PDA. The value of the result returned from `McxSync()` is used to make the determination. Once card connection is confirmed using a function such as `MemCardAccept()` or `_card_info()`, `McxCardType()` can be called. An `McxErrSuccess` result would mean that a PDA was detected, and an `McxErrInvalid` would mean that a Memory Card was detected. However, if the Memory Card connection is determined with `McxErrInvalid`, the operation should be retried to be certain.

Table 18-3: Result value and connection status of `McxSync()`

Value	Macro	Result
0	<code>McxErrSuccess</code>	Normal termination
1	<code>McxErrNoCard</code>	Neither PDA nor Memory Card inserted
2	<code>McxErrInvalid</code>	Communication failure
3	<code>McxErrNewCard</code>	Normal termination (card has been swapped)

This function only registers a process. Check the contents of the buffer in which the results are stored after confirming that the process has completed. Use `McxSync()` to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxSync\(\)](#)

McxCurrCtrl

Control current capacity (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxCurrCtrl (

int port,

int sound,

int infred,

int led)

Port number (see Table 18-1)

1: Disable speaker, 0: Enable speaker

1: Disable transmit, 0: Enable transmit

1: Disable LED, 0: Enable LED

(The meaning of all other values is the same as 1: disabled)

Explanation

Limits the speaker, IR transmission and LED among PDA functions. This is necessary because there is an upper limit to the current that the PlayStation® can supply to the front terminals. Immediately after the PDA is inserted into the PlayStation, all become disabled by default.

The current consumed by each module is indicated in the following table. Since the maximum current that can be supplied by the PlayStation from the two ports totals 160mA, do not exceed this value.

Table 18-4

Module name	Current consumed
CPU chip	10mA
IR module transmission-side	70mA
Speaker	20mA
LED	10mA

Before attempting to use the aforementioned three functions in a PDA application, use Get PDA Status (swi 6) to check the permission status of each function. (For a description of the required processing, please refer to the PDA Kernel Specification document.)

Call McxAllInfo() to check the restrictions on available functions.

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using McxSync().

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxAllInfo\(\)](#), [McxSync\(\)](#)

McxExecApl

Execute a PDA application (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	2/24/99

Syntax

int McxExecApl (

int port,

int aplno,

Port number (see Table 18-1)

Block number of header which contains the desired PDA application to execute.

(Process registration fails if a non-negative value other than 0-15 is specified.)

long arg)

Argument passed to the application to be started.

Explanation

The PDA application with the specified header block number is executed. If the specified block number is not a header block of a PDA application, proper PDA operation cannot be guaranteed. In order to confirm that the target application has started, check the number of the currently executing PDA application by calling `McxGetApl()` or `McxAllInfo()`.

When this function is called from the PDA application, the "PDA Application Exited" flag (bit 11) from the result of Get PDA Status (swi 6) conveys a request to terminate the application from the PlayStation (this flag should be checked periodically). Please see the *PDA Kernel Specification* document for more information on processing required by the PDA application when the application terminates.

This function is used to make a request to terminate the currently executing PDA application. However, if the currently executing application refuses to terminate, `McxGetApl()` should be used, after calling `McxExecApl()`, to ensure that the PDA application has switched.

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using `McxSync()`.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed, or a value other than 0-15 is specified as the block number.)

See note on page 18-4 for more information.

See also

[McxGetApl\(\)](#), [McxAllInfo\(\)](#), [McxSync\(\)](#)

McxExecFlag

Set the PDA application / data ID flag (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	2/24/99

Syntax

int McxExecFlag (

int *port*,

int *block*,

Port number (see Table 18-1)

Block number of the header which contains the desired file for which you wish to set the PDA application flag (The result of dividing the DIRENTRY member head by 64. If the specified value is other than 1-15, process registration fails.)

int *exec*)

1: PDA application flag set, 0: Flag cancelled (All other values, same as 1)

Explanation

Called and set when downloading a PDA application to the PDA. (May be called any time after the file is created.) The PDA application flag is set to 0 when a Memory Card file copy is performed from the PlayStation Memory Card set-up screen, etc.

By referring to the PDA kernel software interrupt (swi 24), the value of the PDA application flag can be used to determine whether or not the PDA application that is executing is an original download or one copied from another PDA and Memory Card.

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using `McxSync()`.

In `libmcx`, issuing only `McxExecFlag()` will not provide an `McxSync()` result of `McxErrInvalid` for a Memory Card. If successful, the PDA application flag will also be set for the corresponding section of the Memory Card which issued this instruction.

If the result from `McxSync()` is `McxErrNewCard`, processing will be interrupted for `McxExecFlag()`. Unverified flags should first be cleared using `MemCardAccept()` and `_card_clear()`, then process registration should be performed again.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed, or a value other than 1-15 was specified for the block number.)

See note on page 18-4 for more information.

See also

[McxSync\(\)](#)

McxFashAcs

Set a PDA application's flash memory write priority (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxFashAcs (

int port,

int mode)

Port number (see Table 18-1)

0: Allows communication with the PlayStation to be temporarily suspended so that the PDA application can write to the flash memory.

1: Disables writing to the flash memory.

(The meaning of all other values is the same as 1: disabled)

Explanation

While a PDA application is writing to the flash memory, performing communication with the PlayStation can lead to processing load and access contention problems. So, when a PDA application is able to write to the flash memory, the PlayStation-side program will first determine whether or not communication with the PDA might fail, then this function can be used to allow the flash memory to be written.

The default is to not allow the PDA application to write to the flash memory. In other words, communication with the PlayStation has priority over writing of the flash memory.

Communication with the PDA is sometimes interrupted while flash memory writing is enabled. Therefore, it sometimes is necessary to perform a retry in order to communicate reliably. Communication with the PDA can be interrupted in the following cases:

When `_card_XXX()` is used and the HwCARD I EvSpTIMOUT event is generated.

When `MemCardXXX()` is used and `McxErCardNotExist` is returned in `*result` of `MemCardSync()`.

When `McxXXX()` is used and `McxErNoCard` is returned in `*result` of `McxSync()`.

Call `McxAllInfo()` to check the restrictions on available functions.

Before attempting to write to the flash memory from a PDA application, use Get PDA Status (swi 6) to check the flash memory write enable status. (For a description of the required processing, please refer to the PDA Kernel Specification document.)

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using `McxSync()`.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxAllInfo\(\)](#), [McxSync\(\)](#)

McxGetApl

Obtain the block number where the executing PDA application is stored (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxGetApl (

int *port*,

int **aplno*)

Port number (see Table 18-1)

Pointer to a memory location which returns the block number where the executing PDA application is stored.

Explanation

The block number of the header where the currently executing PDA application is stored is placed in the memory location pointed to by *aplno*.

If the currently executing application is the "Start-up Application", **aplno* returns 0.

If the currently executing application is other than the "Start-up Application", **aplno* returns the block number of the header where the application is stored.

When *McxAllInfo()* is used, the executing application number together with other information can be obtained during a single frame.

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has completed. Use *McxSync()* to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxExecApl\(\)](#), [McxAllInfo\(\)](#), [McxSync\(\)](#)

McxGetMem

Read the contents of PDA memory (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxGetMem (
int port,           Port number (see Table 18-1)
unsigned char *data, Buffer for storing PDA memory read data
unsigned start,     Read start address
unsigned len)       Read size (bytes), (128 bytes max.)
```

Explanation

The specified number of bytes of PDA memory are read from the specified start address and placed in the buffer pointed to by data.

Process registration fails if the read size exceeds 128 bytes. Registration also fails when the read range extends outside the areas shown below. **Note:** if an attempt is made to access address 0x2***** without specifying virtual flash memory, a bus error will be generated by the PDA.

Readable areas:

```
0x0*****, 0x2*****, 0x4*****, 0x6*****, 0x8*****
0xA*****, 0xB*****, 0xC*****, 0xD*****
```

(* = an arbitrary hex value)

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has completed. Use McxSync() to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed, or the read size exceeded 128 bytes. Also fails when an attempt is made to access an unavailable area of memory.)

See note on page 18-4 for more information.

See also

[McxSetMem\(\)](#), [McxSync\(\)](#)

McxGetSerial

Get the PDA's serial number (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxGetSerial (
  int port,           Port number (see Table 18-1)
  unsigned long *serial)  Pointer to a memory location that will contain the obtained serial number
```

Explanation

Gets the serial number of the PDA.

The serial number, together with other information, can also be obtained using `McxAllInfo()`.

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has completed. Use `McxSync()` to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxAllInfo\(\)](#), [McxGetMem\(\)](#), [McxSync\(\)](#)

McxGetTime

Get the date and time (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxGetTime (
int port,          Port number (see Table 18-1)
unsigned char *time) Buffer for storing the date and time (8 bytes)
```

Explanation

Obtains the date and time from the PDA's real-time clock. The contents of the buffer are shown below. Information other than day of the week are returned as BCD values.

Table 18-5

Offset	0	1	2	3	4	5	6	7
Contents	100 years	Year	Month	Date	Day of week	Hr.	Min.	Sec.

The day of the week values returned in offset 4 are:

Table 18-6

<i>time</i> [4]	0	1	2	3	4	5	6
Day of week	Sun	Mon	Tue	Wed	Thu	Fri	Sat

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has completed. Use `McxSync()` to check for process completion.

Date and time information can be obtained, together with other information, using `McxAllInfo()`.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxAllInfo\(\)](#), [McxSync\(\)](#)

McxGetUIFS

Get user interface status (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxGetUIFS (
  int port,           Port number (see Table 18-1)
  unsigned char *data) Buffer for storing user interface status (8 bytes)
```

Explanation

Obtains user interface status from the PDA. The contents of the buffer are shown below.

Table 18-7

Offset	Contents
0	Alarm time (minutes)
1	Alarm time (hours)
2	(bit)7 RTC set
	(bit)6-4 Area code
	(bit)3-2 Speaker volume
	(bit)1 Key lock
	(bit)0 Alarm
3	Unused
4	Font data start address (LSB)
5	Font data start address (MSB)
6	Unused
7	Unused

- Alarm time of day: 2 BCD digits each.
- RTC set: 0: RTC is unset (data invalid) after reset, 1: RTC set (data valid)
- Area code: 0: Japan, 1: North America, 2: Europe
- Speaker volume: 0: loud, 1: soft, 2: off
- Key lock: 0: unlocked, 1: locked
- Alarm: 0: off, 1: on
- Font data start address: Address relative to 0x4000000

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has completed. Use `McxSync()` to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxSetUIFS\(\)](#), [McxSync\(\)](#)

McxHideTrans

Hide the "data transfer" display (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	2/24/99

Syntax

```
int McxHideTrans (
int port)          Port number (see Table 18-1)
```

Explanation

Hides the "data transfer" display on the LCD screen made visible by McxShowTrans().

When this function is called, a file transfer control callback is generated from the PDA kernel to the currently executing PDA application. This callback, previously registered for "start/stop the data transfer display" using the PDA's "set user callback (swi 1)" terminates the data transfer display on the LCD screen of the PDA. (For a description of the required processing, please refer to the PDA Kernel Specification document.)

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using McxSync().

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxShowTrans\(\)](#), [McxSync\(\)](#)

McxReadDev

Read from the PDA device (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxReadDev (

int *port*,

int *dev*,

unsigned char **param*,

unsigned char **data*)

Port number (see Table 18-1)

Called device number (Reserved devices: 0: RTC read/write, 1: PDA memory read/write, 2: user interface status read/write)

Parameter passed to device

Data read from device

Explanation

Performs a read from a user-defined device or from a reserved device provided on the PDA. In order read from a user-defined device, it is necessary to create a subroutine as specified in \pdadoc\doc\jp\word\pda\kernel.doc (Kernel Service Overview: Communication with the PlayStation : Device Entry Callbacks), then enter it in the device entry table in the Memory Card file header.

This function performs process registration only. The contents of the result buffer can be used after confirming that the process has completed. Use McxSync() to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxWriteDev\(\)](#), [McxSync\(\)](#)

McxSetLED

Switch the LED on/off (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxSetLED (

int *port*,

Port number (see Table 18-1)

int *mode*)

0: Turn LED off, other values: Turn LED on

Explanation

Turns the LED on and off.

McxGetMem() and McxSetMem() can be used to check/set the on/off state of an LED, by directly accessing PIO0 and PIO1.

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using McxSync().

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxGetMem\(\)](#), [McxSetMem\(\)](#), [McxSync\(\)](#)

McxSetMem

Update the contents of PDA memory (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

```
int McxSetMem (  
int port,           Port number (see Table 18-1)  
unsigned char *data, Buffer containing the data to be written to PDA memory  
unsigned start,     Write start address  
unsigned len)       Write size (bytes), (128 bytes max.)
```

Explanation

Overwrites the specified number of bytes of PDA memory starting at the specified address. Process registration fails if the write size exceeds 128 bytes. Registration also fails when the write range extends outside the areas shown below.

Readable areas:

```
0x0*****, 0x6*****, 0xA*****  
0xB*****, 0xC*****, 0xD*****
```

(* = arbitrary hex value)

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using McxSync().

Return value

- 1: Process registration was accepted.
 - 0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed, or the write size exceeded 128 bytes. Also fails when an attempt is made to access an unavailable area of memory.)
- See note on page 18-4 for more information.

See also

[McxGetMem\(\)](#), [McxSync\(\)](#)

McxSetTime

Set the date and time (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxSetTime (
int *port*, Port number (see Table 18-1)
unsigned char **time*) Buffer which contains the date and time to be set (8 bytes)

Explanation

Sets the date and time in the PDA's real-time clock.

In order to set the real-time clock, 150 ms (worst case) are required after communication with the PDA is completed. During this interval, it is not possible to communicate with the PDA. Note that, if an attempt is made to communicate with the PDA during this time, McxSync() will return the McxErrNoCard (Memory Card not inserted) result, even though the PDA is inserted in the Memory Card slot.

The buffer contents are shown below. Information other than day of the week are returned as BCD values.

Table 18-8

Offset	0	1	2	3	4	5	6	7
Contents	100 years	Year	Month	Date	Day of week	Hr.	Min.	Sec.

The relationship between the day of the week and the value returned in offset 4 is as follows:

Table 18-9

<i>time</i> [4]	0	1	2	3	4	5	6
Day of week	Sun	Mon	Tue	Wed	Thu	Fri	Sat

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using McxSync().

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxGetTime\(\)](#), [McxSync\(\)](#)

McxSetUIFS

Set user interface status (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxSetUIFS (

int *port*, Port number (see Table 18-1)

unsigned char **data*) Buffer which contains the specified user interface status (8 bytes)

Explanation

Sets the PDA's user interface status. The contents of the buffer are shown below.

Table 18-10

Offset	Contents
0	Alarm time (minutes)
1	Alarm time (hours)
2	(bit)7 RTC set
	(bit)6-4 Area code
	(bit)3-2 Speaker volume
	(bit)1 Key lock
	(bit)0 Alarm
3	Unused
4	Font data start address (LSB)
5	Font data start address (MSB)
6	Unused
7	Unused

- Alarm time of day: 2 BCD digits each.
- RTC set: 0: RTC is unset (data invalid) after reset, 1: RTC set (data valid)
- Area code: 0: Japan, 1: North America, 2: Europe
- Speaker volume: 0: loud, 1: soft, 2: off
- Key lock: 0: unlocked, 1: locked
- Alarm: 0: off, 1: on
- Font data start address: Address relative to 0x4000000

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using `McxSync()`.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxGetUIFS\(\)](#), [McxSync\(\)](#)

McxShowTrans

Show the "data transfer" display (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	2/24/99

Syntax

```
int McxShowTrans (
  int port,           Port number (see Table 18-1)
  int dir,            Transfer direction. 0: PDA --> PS, other values: PS --> PDA
  int timeout)        Time until the display is cleared if a request to hide the "data transfer" display
                      is not received. (1 second units in the activated application)
```

Explanation

In order to avoid alternate sector processing when saving a PDA application file or a data file that contains a PDA file list icon (i.e., so that the PDA program can be saved to consecutive memory), call this function before opening the file.

Since MemCardFormat() and _card_format() also initialize alternate sectors, do not call McxShowTrans() when formatting the Memory Card (formatting will fail without initializing the alternate sectors).

When this function is called, a file transfer control callback is generated from the PDA kernel to the currently executing PDA application. This callback, previously registered for "start/stop the data transfer display" using the PDA's "set user callback (swi 1)" terminates the data transfer display on the LCD screen of the PDA.

The timeout setting allows the PDA application itself to hide the "data transfer" display, if the PlayStation is reset unexpectedly, etc. Normally, McxHideTrans() is called to hide the display after the file transfer has completed.

(For a description of the required processing, please refer to the PDA Kernel Specification document.)

This function performs process registration only. Before calling another process registration function or a Memory Card access function, check for process completion using McxSync().

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxHideTrans\(\)](#), [McxSync\(\)](#)

McxStartCom

Start the PDA system

Library	Header File	Introduced	Documentation Date
libmcx.lib	libmcx.h	4.4	12/14/98

Syntax

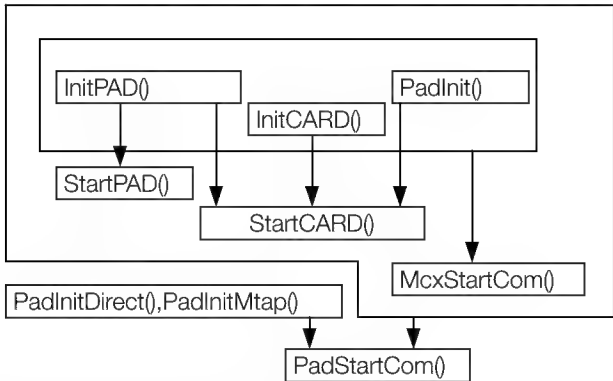
void McxStartCom(void)

Explanation

Starts the PDA system. Enables PDA processing-related interrupts, and allows each of the PDA's process registration functions to be used. The PDA library does not contain any initialization functions (like InitCARD() or MemCardInit()). Instead, the PDA system is activated simply by calling McxStartCom().

McxStartCom() has a restriction in that it must be called in a specific sequence relative to other functions.

For functions connected with arrows, the function at the starting point of the arrows must be called first.



A standard sequence would be as follows.

```
PadInit();
InitPAD();
StartPAD();
InitCARD();
StartCARD();
McxStartCom();
PadInitDirect();
PadStartCom();
```

However, the following three sets of function pairs cannot be called simultaneously:

[PadInit()], [InitPAD(), StartPAD()], [PadInitDirect(), PadStartCom()].

An appropriate set from the three sets should be selected when writing programs.

MemCardInit() and MemCardStart() can be replaced with InitCARD() and StartCARD(). InitPAD() and StartPAD() can be replaced with InitTAP() and StartTAP() or InitGun() and StartGUN().

See also

[McxStopCom\(\)](#)

McxStopCom

Stop the PDA system

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

void McxStopCom(void)

Explanation

Halts PDA system-related interrupts and shuts down the PDA system.

The McxStartCom() and McxStopCom() pair of start/stop functions must be correctly nested with the calling sequence of the start/stop function pairs determined in McxStartCom().

See also

[McxStartCom\(\)](#)

McxSync

Confirm the completion of a registered process

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	2/24/99

Syntax

int McxSync (

int mode, 0: Wait until the registered process completes. 1: Check the current state and return it immediately.

long *cmd, Completed command number

long *result) Registered process result

Explanation

Use McxAllInfo(), McxCurrCtrl(), etc., to check the progress of a registered process. The command number of the executing or completed process is returned in *cmd*. The contents of *cmd* are shown in the following table.

Table 18-11

Value	Macro	Function
1	McxFuncGetApl	McxGetApl()
2	McxFuncExecApl	McxExecApl()
3	McxFuncGetTime	McxGetTime()
4	McxFuncGetMem	McxGetMem()
5	McxFuncSetMem	McxSetMem()
6	McxFuncShowTrans	McxShowTrans()
7	McxFuncHideTrans	McxHideTrans()
8	McxFuncCurrCtrl	McxCurrCtrl()
9	McxFuncSetLED	McxSetLED()
10	McxFuncGetSerial	McxGetSerial()
11	McxFuncExecFlag	McxExecFlag()
12	McxFuncAllInfo	McxAllInfo()
13	McxFuncFlashAcs	McxFlashAcs()
14	McxFuncReadDev	McxReadDev()
15	McxFuncWriteDev	McxWriteDev()
16	McxFuncGetUIFS	McxGetUIFS()
17	McxFuncSetUIFS	McxSetUIFS()
18	McxFuncSetTime	McxSetTime()
19	McxFuncCardType	McxCardType()

The processing result is returned in "result."

Table 18-12

Value	Macro	Result
0	McxErrSuccess	Normal termination
1	McxErrNoCard	Neither PDA nor Memory Card inserted
2	McxErrInvalid	Communication failure
3	McxErrNewCard	Normal termination (card has been swapped)

Return value

Table 18-13

Value	Macro	Process state
0	McxSyncRun	Processing
1	McxSyncFin	Process complete
-1	McxSyncNone	Process unregistered

See also

[McxGetApl\(\)](#), [McxExecApl\(\)](#), [McxGetTime\(\)](#), [McxGetMem\(\)](#), [McxSetMem\(\)](#), [McxShowTrans\(\)](#), [McxHideTrans\(\)](#), [McxCurrCtrl\(\)](#), [McxFlashAcs\(\)](#), [McxSetLED\(\)](#), [McxGetSerial\(\)](#), [McxExecFlag\(\)](#), [McxAllInfo\(\)](#)

McxWriteDev

Write to a PDA device (process registration)

Library	Header File	Introduced	Documentation Date
<i>libmcx.lib</i>	<i>libmcx.h</i>	4.4	12/14/98

Syntax

int McxReadDev (

int *port*,

int *dev*,

unsigned char **param*,

unsigned char **data*)

Port number (see Table 18-1)

Called device number (Reserved devices: 0: RTC read/write, 1: PDA memory read/write, 2: user interface status read/write)

Parameter passed to device

Data written to device

Explanation

Performs a write to a user-defined device or to a reserved device provided on the PDA. In order to write to a user-defined device, it is necessary to create a subroutine as specified in the PDA Kernel Specification document (Kernel Services Overview - Device Entry Callback section of "Communication with the PlayStation"), then enter it in the device entry table in the Memory Card file header.

This function only registers a process. Check the contents of the buffer in which the results are stored after confirming that the process has completed. Use `McxSync()` to check for process completion.

Return value

1: Process registration was accepted.

0: Process registration failed. (An attempt was made to register a new process before the previously registered process completed.)

See note on page 18-4 for more information.

See also

[McxReadDev\(\)](#), [McxSync\(\)](#)

Chapter 19:Memory Card GUI Module (mcgui)

Table of Contents

Structures

McGuiEnv	19-3
sMcGuiBg	19-4
sMcGuiCards	19-5
sMcGuiController	19-6
sMcGuiCursor	19-7
sMcGuiSnd	19-8
sMcGuiTexture	19-9

Functions

McGuiLoad	19-10
McGuiSave	19-11
McGuiSetExternalFont	19-12

McGuiEnv

Memory Card GUI module structure

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Structure

```
struct McGuiEnv {
    sMcGuiCards cards;           Memory Card data structure
    sMcGuiBg bg;                 BG data structure
    sMcGuiController controller; Controller-related data structure
    sMcGuiSnd sound;             BGM and sound effect data structure
    sMcGuiTexture texture;       Texture data structure
    sMcGuiCursor cursor;         Cursor data structure
};
```

Explanation

This is the main structure used by the memory card GUI module.
Each member is a separate structure.

sMcGuiBg

BG data structure

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Structure

```
struct sMcGuiBg {
    short mode;
    signed char scrollDirect;
    signed char scrollSpeed;
    u_long* timadr;
};
```

BG mode
 0: Scroll mode Use the internal 64 X 64 tiled-texture scroll mode
 1: Use the texture specified by the still-image mode (timadr)
 Scroll direction (valid when BG mode = 0)
 0: up, 1: top left, 2: left, 3: bottom left 4: down, 5: bottom right, 6:
 right, 7: top right
 Scroll speed (valid when BG mode = 0)
 0: Stopped 1: 1/60 sec., 2: 1/30 sec., 3: 1/20 sec.
 Header address of TIM data for BG (valid when BG mode = 1)

Explanation

Part of the [McGuiEnv](#) structure.

sMcGuiCards

Memory Card data structure

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Structure

struct sMcGuiCards {	
char <i>file</i> [21];	File name Use only ASCII except for 0x00, 0x2a(*), and 0x3f(?). The string is terminated by 0x00.
char <i>title</i> [65];	Document name Only full-size 32 characters of (SJIS) non-kanji and level-1 kanji. However, 0x84bf to 0x889e cannot be used. The string is terminated by 0x00.
char <i>frame</i> ;	Reserved area (unusable)
char <i>block</i> ;	Number of icon images (automatically animated) 1-3
u_long* <i>iconAddr</i> ;	TIM data header address for icon image
u_long* <i>dataAddr</i> ;	Header Address of game data
long <i>dataBytes</i> ;	Number of game data bytes (128-byte units)
};	Number of game data blocks (1-15)

Explanation

This structure holds information used for loading and saving game data.

Part of the [McGuiEnv](#) structure.

sMcGuiController

Controller data structure

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Structure

```
struct sMcGuiController {
    volatile u_char* buf[2];           Controller's receive data buffer
    struct {
        int flag;
        u_long BUTTON_OK;
        u_long BUTTON_CANCEL;
    } type1;                           Default controller
    struct {
        int flag;
        u_long BUTTON_OK;
        u_long BUTTON_CANCEL;
    } type2;                           Mouse
    struct {
        int flag;
        u_long BUTTON_OK;
        u_long BUTTON_CANCEL;
    } type3;                           Analog joystick, DUAL SHOCK
    struct {
        int flag;
        u_long BUTTON_OK;
        u_long BUTTON_CANCEL;
    } type4;                           NeGcon
};
```

Explanation

This structure holds controller information. It is part of the [McGuiEnv](#) structure.

Set the *flag* members of the supported controller types to 1, and set the respective button codes in the *BUTTON_OK/BUTTON_CANCEL* fields. Set the *flag* members of unsupported controller types to 0.

sMcGuiCursor

Cursor data structure

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Structure

```
struct sMcGuiCursor {
    char type;           Cursor shape (not supported)
    u_char r;            Color code (0-255)
    u_char g;            Color code (0-255)
    u_char b;            Color code (0-255)
};
```

Explanation

This structure sets the color and shape of the menu cursor. It is part of the [McGuiEnv](#) structure.

Note: *type* is currently not supported.

sMcGuiSnd

BGM and sound effects data structure

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Structure

```

struct sMcGuiSnd {
    int MVOL;           Main volume (0-127)
    int isReverb;       0: reverb OFF, 1: reverb ON
    int reverbType;     Reverb type
    int reverbDepth;    Reverb depth (0-127)
    struct {
        int isbgm;      0: no BGM, 1:BGM
        u_long* seq;    Header address of SEQ data
        u_long* vh;     Header address of VH data for SEQ
        u_long* vb;     Header address of VB data for SEQ
        int SVOL;       SEQ volume (0-127)
    } bgm;
    struct {
        int isse;       0: no sound effects, 1: sound effects
        u_long* vh;     Header address of VH data for SE
        u_long* vb;     Header address of VB data for SE
        int vol;        Sound effects volume
        int prog;       Sound effects program number
        int TONE_OK;    Tone number of confirmation/execution tone
        int TONE_CANCEL; Tone number of CANCEL tone
        int TONE_CURSOR; Tone number during cursor operation
        int TONE_ERROR; Tone number of error tone
    } se;
};

```

Explanation

This structure contains information about the BGM that plays on the Memory Card screen and the sound effect that is produced when the cursor is moved, etc.

sMcGuiTexture

Texture data structure

Module	Header File	Introduced	Documentation Date
mcgui.obj	mcgui.h	4.4	12/14/98

Structure

```
struct sMcGuiTexture {  
    u_long* addr;           Header address of TIM data  
};
```

Explanation

This structure specifies the texture data used internally by the module. It is part of the [McGuiEnv](#) structure.

Note: For format information, refer to the *Run-time Library Overview*.

McGuiLoad

Load game data

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Syntax

```
int McGuiLoad(  
McGuiEnv * env)           Memory Card GUI module structure
```

Explanation

Invokes the load operation of the Memory Card screen.

Terminates when the load has completed or when the cancel button is clicked on the Slot Selection screen.

Return value

1 after the load operation completes

0 if the cancel button was used to terminate the load

-1 if an invalid value is specified in the [McGuiEnv](#) structure

McGuiSave

Save game data

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.4	12/14/98

Syntax

**int McGuiSave(
[McGuiEnv](#) *env)** Memory Card GUI module structure

Explanation

Invokes the save operation of the Memory Card screen.
Terminates when the save is completed or when the cancel button is clicked on the Slot Selection screen.

Return value

- 1 after the save operation completes.
- 0 if the cancel button was used to terminate the save.
- 1 if an invalid value is specified in the [McGuiEnv](#) structure.

McGuiSetExternalFont

Sets English mode

Module	Header File	Introduced	Documentation Date
<i>mcgui.obj</i>	<i>mcgui.h</i>	4.5	2/24/99

Syntax

int McGuiSetExternalFont(

McGuiEnv *env,

Memory Card GUI module structure

int mode)

Language mode (always 1)

Explanation

Switches the messages used on the Memory Card screen to English. For display in Japanese, this function should not be executed.

Index

Structures

—
_GsFCALL (libgs), 1-27
_GsPOSITION (libgs), 1-30
_SsFCALL (libsnd), 14-12

C

CdlAtv (libcd), 10-3
CdlFILE (libcd), 10-4
CdlFILTER (libcd), 10-5
CdlLOC (libcd), 10-6
CRVECTOR3 (libgte), 8-5
CRVECTOR4 (libgte), 8-6
CVECTOR (libgte), 8-7

D

DECDCTENV (libpress), 6-3
DIRENTRY (libapi), 1-3
DISPENV (libgpu), 1-5
DIVPOLYGON3 (libgte), 8-8
DIVPOLYGON4 (libgte), 8-9
DR_AREA (libgpu), 1-7
DR_ENV (libgpu), 1-8
DR_LOAD (libgpu), 1-9
DR_MODE (libgpu), 1-10
DR_MOVE (libgpu), 1-11
DR_OFFSET (libgpu), 1-12
DR_STP (libgpu), 1-13
DR_TPAGE (libgpu), 1-14
DR_TWIN (libgpu), 1-15
DRAWENV (libgpu), 1-6
DslATV (libds), 1-4
DslFILE (libds), 1-5
DslFILTER (libds), 1-6
DslLOC (libds), 1-7
DVECTOR (libgte), 8-10

E

ENCSPUENV (libpress), 6-4
EvCB (libapi), 1-4
EVECTOR (libgte), 8-11
EXEC (libapi), 1-5

G

GsARGUNIT (libhmd), 1-3
GsARGUNIT_ANIM (libhmd), 1-4
GsARGUNIT_GND... (libhmd), 1-5
GsARGUNIT_IMAGE (libhmd), 1-6
GsARGUNIT_JntMIME (libhmd), 1-7
GsARGUNIT_NORMAL (libhmd), 1-8
GsARGUNIT_RstJntMIME (libhmd), 1-9
GsARGUNIT_RstVNMIME (libhmd), 1-10
GsARGUNIT_SHARED (libhmd), 1-11
GsARGUNIT_VNMIME (libhmd), 1-12
GsBG (libgs), 1-3
GsBOXF (libgs), 1-4
GsCELL (libgs), 1-5
GsCOORD2PARAM (libgs), 1-6
GsCOORDINATE2 (libgs), 1-7
GsCOORDUNIT (libhmd), 1-13

GsDOBJ2 (libgs), 1-8
GsDOBJ3 (libgs), 1-10
GsDOBJ5 (libgs), 1-11
GsF_LIGHT (libgs), 1-14
GsFOGPARAM (libgs), 1-13
GsGLINE (libgs), 1-15
GsIMAGE (libgs), 1-16
GsLINE (libgs), 1-17
GsMAP (libgs), 1-18
GsOBJTABLE2 (libgs), 1-19
GsOT (libgs), 1-20
GsOT_TAG (libgs), 1-21
GsRVIEW2 (libgs), 1-22
GsRVIEWUNIT (libhmd), 1-14
GsSEH (libhmd), 1-15
GsSEQ (libhmd), 1-16
GsSPRITE (libgs), 1-23
GsTYPEUNIT (libhmd), 1-18
GsUNIT (libhmd), 1-20
GsVIEW2 (libgs), 1-25
GsVIEWUNIT (libhmd), 1-21
GsWORKUNIT (libhmd), 1-22

L

LINE_F2, LINE_F3, LINE_F4 (libgpu), 1-16
LINE_G2, LINE_G3, LINE_G4 (libgpu), 1-17

M

MATRIX (libgte), 8-12

P

POL3 (libgte), 8-13
POL4 (libgte), 8-14
POLY_F3, POLY_F4 (libgpu), 1-19
POLY_FT3, POLY_FT4 (libgpu), 1-20
POLY_G3, POLY_G4 (libgpu), 1-22
POLY_GT3, POLY_GT4 (libgpu), 1-23
ProgAtr (libsnd), 14-5

Q

QMESH (libgte), 8-15

R

RECT (libgpu), 1-25
RECT32 (libgpu), 1-26
RVECTOR (libgte), 8-16

S

SndRegisterAttr (libsnd), 14-6
SndVoiceStats (libsnd), 14-7
SndVolume (libsnd), 14-8
SndVolume2 (libsnd), 14-9
SPOL (libgte), 8-17
SPRT (libgpu), 1-27
SPRT_8, SPRT_16 (libgpu), 1-28
SpuCommonAttr (libspu), 15-5
SpuDecodeData (libspu), 15-6
SpuEnv (libspu), 15-7
SpuExtAttr (libspu), 15-8

SpuVoiceAttr (libspu), 15-9
SpuReverbAttr (libspu), 15-10
SpuStEnv (libspu), 15-11
SpuStVoiceAttr (libspu), 15-12
SpuVoiceAttr (libspu), 15-13
SpuVolume (libspu), 15-14
StHEADER (libbcd), 10-7
SVECTOR (libgte), 8-18

T

TCB (libapi), 1-6
TCBH (libapi), 1-7

Functions

—
_96_init (libapi), 1-78
_96_remove (libapi), 1-79
_boot (libapi), 1-80
_bu_init (libcard), 4-6
_card_auto (libcard), 4-7
_card_chan (libcard), 4-8
_card_clear (libcard), 4-9
_card_format (libcard), 4-10
_card_info (libcard), 4-11
_card_load (libcard), 4-12
_card_read (libcard), 4-13
_card_status (libcard), 4-14
_card_wait (libcard), 4-15
_card_write (libcard), 4-16
_comb_control (libcomb), 13-3
_get_erno (libapi), 1-81
_get_error (libapi), 1-82
_new_card (libcard), 4-17
_sio_control (libsio), 16-6

A

abs (libc/libc2), 2-3
acos (libmath), 3-3
AddCOMB (libcomb), 13-5
AddPrim (libgpu), 1-33
AddPrims (libgpu), 1-34
AddSIO (libsio), 16-3
ApplyMatrix (libgte), 8-21
ApplyMatrixLV (libgte), 8-22
ApplyMatrixSV (libgte), 8-23
ApplyRotMatrix (libgte), 8-24
ApplyRotMatrixLV (libgte), 8-25
ApplyTransposeMatrixLV (libgte), 8-26
asin (libmath), 3-4
atan (libmath), 3-5
atan2 (libmath), 3-6
atof (libmath), 3-7
atoi (libc/libc2), 2-4
atol (libc/libc2), 2-5
AverageZ3 (libgte), 8-27
AverageZ4 (libgte), 8-28

B

bcmp (libc/libc2), 2-6
bcopy (libc/libc2), 2-7
BreakDraw (libgpu), 1-35
bsearch (libc/libc2), 2-8
bzero (libc/libc2), 2-9

TILE (libgpu), 1-29
TILE_1, TILE_8, TILE_16 (libgpu), 1-30
TIM_IMAGE (libgpu), 1-31
TMD_PRIM (libgpu), 1-32
TMD_STRUCT (libgs), 1-26
TMESH (libgte), 8-19
ToT (libapi), 1-8

V

VabHdr (libsnd), 14-10
VagAtr (libsnd), 14-11
VECTOR (libgte), 8-20

C

calloc (libc/libc2), 2-10
calloc2 (libapi), 1-9
calloc3 (libapi), 1-10
catan (libgte), 8-29
CatPrim (libgpu), 1-36
ccos (libgte), 8-30
cd (libapi), 1-11
CdComstr (libbcd), 10-8
CdControl (libbcd), 10-9
CdControlB (libbcd), 10-11
CdControlF (libbcd), 10-12
CdDataCallback (libbcd), 10-13
CdDataSync (libbcd), 10-14
CdDiskReady (libbcd), 10-15
CdFlush (libbcd), 10-16
CdGetDiskType (libbcd), 10-17
CdGetSector (libbcd), 10-18
CdGetSector2 (libbcd), 10-19
CdGetToc (libbcd), 10-20
CdInit (libbcd), 10-21
CdIntstr (libbcd), 10-22
CdIntToPos (libbcd), 10-23
CdLastCom (libbcd), 10-24
CdLastPos (libbcd), 10-25
CdMix (libbcd), 10-26
CdMode (libbcd), 10-27
CdPlay (libbcd), 10-28
CdPosToInt (libbcd), 10-29
CdRead (libbcd), 10-30
CdRead2 (libbcd), 10-31
CdReadBreak (libbcd), 10-32
CdReadCallback (libbcd), 10-33
CdReadExec (libbcd), 10-34
CdReadFile (libbcd), 10-35
CdReadSync (libbcd), 10-36
CdReady (libbcd), 10-37
CdReadyCallback (libbcd), 10-38
CdReset (libbcd), 10-39
CdSearchFile (libbcd), 10-40
CdSetDebug (libbcd), 10-41
CdStatus (libbcd), 10-42
CdSync (libbcd), 10-43
CdSyncCallback (libbcd), 10-44
ceil (libmath), 3-8
ChangeClearPAD (libapi), 1-12
ChangeClearSIO (libcomb), 13-6
ChangeTh (libapi), 1-13
CheckCallback (libetc), 12-3
CheckPrim (libgpu), 1-37
ClearImage (libgpu), 1-38
ClearImage2 (libgpu), 1-39
ClearOTag (libgpu), 1-40
ClearOTagR (libgpu), 1-41

Clip3F (libgte), 8-31, 8-33
 Clip3FP (libgte), 8-31, 8-33
 Clip3FT (libgte), 8-31, 8-33
 Clip3FTP (libgte), 8-31, 8-33
 Clip3G (libgte), 8-31, 8-33
 Clip3GP (libgte), 8-31, 8-33
 Clip3GT (libgte), 8-31, 8-33
 Clip3GTP (libgte), 8-31, 8-33
 cIn (libgte), 8-35
 close (libapi), 1-14
 CloseEvent (libapi), 1-15
 CloseTh (libapi), 1-16
 ColorCol (libgte), 8-36
 ColorDpq (libgte), 8-37
 ColorMatCol (libgte), 8-38
 ColorMatDpq (libgte), 8-39
 CompMatrix (libgte), 8-40
 CompMatrixLV (libgte), 8-41
 ContinueDraw (libgpu), 1-42
 cos (libmath), 3-9
 cosh (libmath), 3-10
 csin (libgte), 8-42
 csqrt (libgte), 8-43

D

DecDCTBufSize (libpress), 6-5
 DecDCTGetEnv (libpress), 6-6
 DecDCTIn (libpress), 6-7
 DecDCTInCallback (libpress), 6-8
 DecDCTInSync (libpress), 6-9
 DecDCTOut (libpress), 6-10
 DecDCTOutCallBack (libpress), 6-11
 DecDCTOutSync (libpress), 6-12
 DecDCTPutEnv (libpress), 6-13
 DecDCTReset (libpress), 6-14
 DecDCTvic (libpress), 6-15
 DecDCTvic2 (libpress), 6-16
 DecDCTvicBuild (libpress), 6-17
 DecDCTvicSize (libpress), 6-18
 DecDCTvicSize2 (libpress), 6-19
 DelCOMB (libcomb), 13-7
 delete (libapi), 1-23
 DeliverEvent (libapi), 1-17
 DelSIO (libsio), 16-4
 DisableEvent (libapi), 1-18
 DisablePAD (libapi), 1-19
 DisableTAP (libtap), 12-4
 DivideF3 (libgte), 8-44
 DivideF4 (libgte), 8-44
 DivideFT3 (libgte), 8-44
 DivideFT4 (libgte), 8-44
 DivideG3 (libgte), 8-44
 DivideG4 (libgte), 8-44
 DivideGT3 (libgte), 8-44
 DivideGT4 (libgte), 8-44
 dmy_Ss... (libsnd), 14-14
 dmyGsPrst... (libgs), 1-31
 dmyGsTMD... (libgs), 1-32
 DpqColor (libgte), 8-46
 DpqColor3 (libgte), 8-47
 DpqColorLight (libgte), 8-48
 DrawOTag (libgpu), 1-43
 DrawOTag2 (libgpu), 1-44
 DrawOTagEnv (libgpu), 1-45
 DrawOTagLO (libgpu), 1-46
 DrawPrim (libgpu), 1-47
 DrawSync (libgpu), 1-48
 DrawSyncCallback (libgpu), 1-49
 DsClose (libds), 1-8
 DsCommand (libds), 1-9
 DsComstr (libds), 1-10

DsControl (libds), 1-11
 DsControlB (libds), 1-12
 DsControlF (libds), 1-13
 DsDataCallback (libds), 1-14
 DsDataSync (libds), 1-15
 DsEndReadySystem (libds), 1-16
 DsFlush (libds), 1-17
 DsGetDiskType (libds), 1-18
 DsGetSector (libds), 1-19
 DsGetSector2 (libds), 1-20
 DsGetToc (libds), 1-21
 DsInit (libds), 1-22
 DsInstr (libds), 1-23
 DsIntToPos (libds), 1-24
 DsLastCom (libds), 1-25
 DsLastPos (libds), 1-26
 DsMix (libds), 1-27
 DsPacket (libds), 1-28
 DsPlay (libds), 1-29
 DsPosToInt (libds), 1-30
 DsQueueLen (libds), 1-31
 DsRead (libds), 1-32
 DsRead2 (libds), 1-33
 DsReadBreak (libds), 1-34
 DsReadCallback (libds), 1-35
 DsReadExec (libds), 1-36
 DsReadFile (libds), 1-37
 DsReadSync (libds), 1-38
 DsReady (libds), 1-39
 DsReadyCallback (libds), 1-40
 DsReadySystemMode (libds), 1-41
 DsReset (libds), 1-42
 DsSearchFile (libds), 1-43
 DsSetDebug (libds), 1-44
 DsShellOpen (libds), 1-45
 DsStartReadySystem (libds), 1-46
 DsStatus (libds), 1-47
 DsSync (libds), 1-48
 DsSyncCallback (libds), 1-49
 DsSystemStatus (libds), 1-50
 DumpClut (libgpu), 1-50
 DumpDispEnv (libgpu), 1-51
 DumpDrawEnv (libgpu), 1-52
 DumpOTag (libgpu), 1-53
 DumpTPage (libgpu), 1-54

E

EigenMatrix (libgte), 8-49
 EnableEvent (libapi), 1-20
 EnablePAD (libapi), 1-21
 EnableTAP (libtap), 12-5
 EncSPU (libpress), 6-20
 EncSPU2 (libpress), 6-22
 EnterCriticalSection (libapi), 1-22
 erase (libapi), 1-23
 Exception (libapi), 1-24
 Exec (libapi), 1-25
 exit (libc/libc2), 2-11
 ExitCriticalSection (libapi), 1-26
 exp (libmath), 3-11

F

firstfile (libapi), 1-27
 floor (libmath), 3-13
 FlushCache (libapi), 1-28
 fmod (libmath), 3-14
 FntFlush (libgpu), 1-55
 FntLoad (libgpu), 1-56
 FntOpen (libgpu), 1-57
 FntPrint (libgpu), 1-58

4 XXX Library Functions

format (libapi), 1-29
free (libc/libc2), 2-12
free2 (libapi), 1-30
free3 (libapi), 1-31
frexp (libmath), 3-15

G

getc (libc/libc2), 2-13
getchar (libc/libc2), 2-14
GetClut (libgpu), 1-59
GetConf (libapi), 1-32
GetCr (libapi), 1-33
GetDispEnv (libgpu), 1-60
GetDrawArea (libgpu), 1-61
GetDrawEnv (libgpu), 1-62
GetDrawEnv2 (libgpu), 1-63
GetDrawMode (libgpu), 1-64
GetDrawOffset (libgpu), 1-65
GetGp (libapi), 1-34
GetGraphDebug (libgpu), 1-66
GetODE (libgpu), 1-67
GetRCnt (libapi), 1-35
gets (libc/libc2), 2-15
GetSp (libapi), 1-36
GetSr (libapi), 1-37
GetSysSp (libapi), 1-38
GetTexWindow (libgpu), 1-68
GetTimSize (libgpu), 1-69
GetTPage (libgpu), 1-70
GetVideoMode (libetc), 12-6
GsA4div... (libgs), 1-33
GsClearOt (libgs), 1-36
GsClearVcount (libgs), 1-37
GsCutOt (libgs), 1-38
GsDefDispBuff (libgs), 1-39
GsDefDispBuff2 (libgs), 1-40
GsDrawOt (libgs), 1-41
GsDrawOtIO (libgs), 1-42
GsGetActiveBuffer (libgs), 1-43
GsGetHeadpUnit (libhmd), 1-23
GsGetLs (libgs), 1-44
GsGetLsUnit (libhmd), 1-24
GsGetLw (libgs), 1-45
GsGetLws (libgs), 1-46
GsGetLwsUnit (libhmd), 1-25
GsGetLwUnit (libhmd), 1-26
GsGetTimInfo (libgs), 1-47
GsGetVcount (libgs), 1-48
GsGetWorkBase (libgs), 1-49
GsInit3D (libgs), 1-50
GsInitCoordinate2 (libgs), 1-51
GsInitFixBg16 (libgs), 1-52
GsInitFixBg32 (libgs), 1-52
GsInitGraph (libgs), 1-53
GsInitGraph2 (libgs), 1-54
GsInitRstNrmMIME (libhmd), 1-27
GsInitRstVtxMIME (libhmd), 1-28
GsInitVcount (libgs), 1-55
GsLinkAnim (libhmd), 1-29
GsLinkObject3 (libgs), 1-56
GsLinkObject4 (libgs), 1-57
GsLinkObject5 (libgs), 1-58
GsMapCoordUnit (libhmd), 1-30
GsMapModelingData (libgs), 1-59
GsMapUnit (libhmd), 1-31
GsMulCoord0 (libgs), 1-60
GsMulCoord2 (libgs), 1-61
GsMulCoord3 (libgs), 1-62
GsPresetObject (libgs), 1-63
GsPrst... (libgs), 1-64
GsScaleScreen (libgs), 1-66

GsScanAnim (libhmd), 1-32
GsScanUnit (libhmd), 1-33
GsSetAmbient (libgs), 1-67
GsSetClip (libgs), 1-68
GsSetClip2 (libgs), 1-69
GsSetClip2D (libgs), 1-70
GsSetDrawBuffClip (libgs), 1-71
GsSetDrawBuffOffset (libgs), 1-72
GsSetFlatLight (libgs), 1-73
GsSetFogParam (libgs), 1-74
GsSetLightMatrix (libgs), 1-75
GsSetLightMatrix2 (libgs), 1-76
GsSetLightMode (libgs), 1-77
GsSetLsMatrix (libgs), 1-78
GsSetOffset (libgs), 1-79
GsSetOrign (libgs), 1-80
GsSetProjection (libgs), 1-81
GsSetRefView2 (libgs), 1-82
GsSetRefView2L (libgs), 1-83
GsSetRefViewLUnit (libhmd), 1-34
GsSetRefViewUnit (libhmd), 1-35
GsSetView2 (libgs), 1-84
GsSetViewUnit (libhmd), 1-36
GsSetWorkBase (libgs), 1-85
GsSortBg (libgs), 1-86
GsSortBoxFill (libgs), 1-87
GsSortClear (libgs), 1-88
GsSortFastBg (libgs), 1-86
GsSortFastSprite (libgs), 1-98
GsSortFixBg16 (libgs), 1-89
GsSortFixBg32 (libgs), 1-89
GsSortFlipSprite (libgs), 1-98
GsSortGLine (libgs), 1-90
GsSortLine (libgs), 1-90
GsSortObject3 (libgs), 1-91
GsSortObject4 (libgs), 1-92
GsSortObject4J (libgs), 1-93
GsSortObject5 (libgs), 1-94
GsSortObject5J (libgs), 1-95
GsSortOt (libgs), 1-96
GsSortPoly (libgs), 1-97
GsSortSprite (libgs), 1-98
GsSortUnit (libhmd), 1-37
GsSwapDispBuffer (libgs), 1-99
GsTMDdiv... (libgs), 1-100
GsTMDfast... (libgs), 1-104
GsTMDfastN... (libgs), 1-104
GsU... (libhmd), 1-38
GsU_03000000 (libhmd), 1-39
GsU_03000001... (libhmd), 1-41
GsU_03010110... (libhmd), 1-43
GsU_040100... (libhmd), 1-45
gteMIMEfunc (libgte), 8-50

H

hypot (libmath), 3-16

I

InitCARD (libcard), 4-3
InitClip (libgte), 8-51
InitGeom (libgte), 8-52
InitGUN (libgun), 12-7
InitHeap (libapi), 1-39
InitHeap2 (libapi), 1-40
InitHeap3 (libapi), 1-41
InitPAD (libapi), 1-42
InitTAP (libtap), 12-8
Intpl (libgte), 8-53
InvSquareRoot (libgte), 8-54
ioctl (libapi), 1-43

lseek (libapi), 1-49
 IsEndPrim (libgpu), 1-71
 IsIdleGPU (libgpu), 1-72
 IsIdMatrix (libgte), 8-55

K

KanjiFntClose (libgpu), 1-73
 KanjiFntFlush (libgpu), 1-74
 KanjiFntOpen (libgpu), 1-75
 KanjiFntPrint (libgpu), 1-76
 Krom2RawAdd (libapi), 1-44
 Krom2RawAdd2 (libapi), 1-45
 Krom2Tim (libgpu), 1-77

L

labs (libc/libc2), 2-17
 ldexp (libmath), 3-17
 LightColor (libgte), 8-56
 Load (libapi), 1-46
 LoadAverage0 (libgte), 8-57
 LoadAverage12 (libgte), 8-58
 LoadAverageByte (libgte), 8-59
 LoadAverageCol (libgte), 8-60
 LoadAverageShort0 (libgte), 8-61
 LoadAverageShort12 (libgte), 8-62
 LoadClut (libgpu), 1-78
 LoadClut2 (libgpu), 1-79
 LoadExec (libapi), 1-47
 LoadImage (libgpu), 1-80
 LoadImage2 (libgpu), 1-81
 LoadTest (libapi), 1-48
 LoadTPage (libgpu), 1-82
 LocalLight (libgte), 8-63
 log (libmath), 3-18
 log10 (libmath), 3-19
 longjmp (libc/libc2), 2-18
 Lzc (libgte), 8-64

M

malloc (libc/libc2), 2-19
 malloc2 (libapi), 1-50
 malloc3 (libapi), 1-51
 MargePrim (libgpu), 1-83
 MatrixNormal (libgte), 8-65
 MatrixNormal_0 (libgte), 8-66
 MatrixNormal_1 (libgte), 8-67
 MatrixNormal_2 (libgte), 8-68
 MemCardAccept (libmcrd), 5-3
 MemCardCallback (libmcrd), 5-4
 MemCardClose (libmcrd), 5-5
 MemCardCreateFile (libmcrd), 5-6
 MemCardDeleteFile (libmcrd), 5-7
 MemCardEnd (libmcrd), 5-8
 MemCardExist (libmcrd), 5-9
 MemCardFormat (libmcrd), 5-10
 MemCardGetDirent (libmcrd), 5-11
 MemCardInit (libmcrd), 5-12
 MemCardOpen (libmcrd), 5-13
 MemCardReadData (libmcrd), 5-14
 MemCardReadFile (libmcrd), 5-15
 MemCardStart (libmcrd), 5-16
 MemCardStop (libmcrd), 5-17
 MemCardSync (libmcrd), 5-18, 5-19
 MemCardWriteData (libmcrd), 5-20
 MemCardWriteFile (libmcrd), 5-21
 memchr (libc/libc2), 2-20
 memcmp (libc/libc2), 2-21
 memcpy (libc/libc2), 2-22
 memmove (libc/libc2), 2-23

memset (libc/libc2), 2-24
 modf (libmath), 3-20
 MoveImage (libgpu), 1-84
 MoveImage2 (libgpu), 1-85
 MulMatrix (libgte), 8-69
 MulMatrix0 (libgte), 8-70
 MulMatrix2 (libgte), 8-71
 MulRotMatrix (libgte), 8-72
 MulRotMatrix0 (libgte), 8-73

N

nextfile (libapi), 1-52
 NextPrim (libgpu), 1-86
 NormalClip (libgte), 8-74
 NormalColor (libgte), 8-75
 NormalColor_nom (libgte), 8-75
 NormalColor3 (libgte), 8-76
 NormalColor3_nom (libgte), 8-76
 NormalColorCol (libgte), 8-77
 NormalColorCol_nom (libgte), 8-77
 NormalColorCol3 (libgte), 8-78
 NormalColorCol3_nom (libgte), 8-78
 NormalColorDpq (libgte), 8-79
 NormalColorDpq_nom (libgte), 8-79
 NormalColorDpq3 (libgte), 8-80
 NormalColorDpq3_nom (libgte), 8-80

O

open (libapi), 1-53
 OpenEvent (libapi), 1-54
 OpenTh (libapi), 1-55
 OpenTIM (libgpu), 1-87
 OpenTMD (libgpu), 1-88
 otz2p (libgte), 8-81
 OuterProduct0 (libgte), 8-82
 OuterProduct12 (libgte), 8-83

P

p2otz (libgte), 8-84
 PadChkVsync (libpad), 12-9
 PadEnableCom (libpad), 12-10
 PadEnableGun (libpad), 12-11
 PadGetState (libpad), 12-12
 PadInfoAct (libpad), 12-13
 PadInfoComb (libpad), 12-15
 PadInfoMode (libpad), 12-16
 PadInit (libetc), 12-18
 PadInitDirect (libpad), 12-19
 PadInitGun (libpad), 12-20
 PadInitMtap (libpad), 12-22
 PadRead (libetc), 12-23
 PadRemoveGun (libpad), 12-24
 PadSetAct (libpad), 12-25
 PadSetActAlign (libpad), 12-27
 PadSetMainMode (libpad), 12-28
 PadStartCom (libpad), 12-29
 PadStop (libetc), 12-30
 PadStopCom (libpad), 12-31
 pers_map (libgte), 8-85
 PhongLine (libgte), 8-86
 PopMatrix (libgte), 8-87
 pow (libmath), 3-21
 printf (libc/libc2), 2-25
 printf2 (libmath), 3-22
 PushMatrix (libgte), 8-88
 putc (libc/libc2), 2-26
 putchar (libc/libc2), 2-27
 PutDispEnv (libgpu), 1-89
 PutDrawEnv (libgpu), 1-90

puts (libc/libc2), 2-28

Q

qsort (libc/libc2), 2-29

R

rand (libc/libc2), 2-30
 ratan2 (libgte), 8-89
 rcos (libgte), 8-90
 RCpolyF3 (libgte), 8-91
 RcpolyF4 (libgte), 8-92
 RCpolyFT3 (libgte), 8-91
 RcpolyFT4 (libgte), 8-92
 RCpolyG3 (libgte), 8-91
 RcpolyG4 (libgte), 8-92
 RCpolyGT3 (libgte), 8-91
 RcpolyGT4 (libgte), 8-92
 read (libapi), 1-56
 ReadColorMatrix (libgte), 8-93
 ReadGeomOffset (libgte), 8-94
 ReadGeomScreen (libgte), 8-95
 ReadLightMatrix (libgte), 8-96
 ReadRGBBifo (libgte), 8-97
 ReadRotMatrix (libgte), 8-98
 ReadSXSyifo (libgte), 8-99
 ReadSZfifo3 (libgte), 8-100
 ReadSZfifo4 (libgte), 8-101
 ReadTIM (libgpu), 1-91
 ReadTMD (libgpu), 1-92
 realloc (libc/libc2), 2-31
 realloc2 (libapi), 1-57
 realloc3 (libapi), 1-58
 RemoveGUN (libgun), 12-32
 rename (libapi), 1-59
 ResetCallback (libetc), 12-33
 ResetGraph (libgpu), 1-93
 ResetRCnt (libapi), 1-60
 RestartCallback (libetc), 12-34
 ReturnFromException (libapi), 1-61
 RotAverage3 (libgte), 8-102
 RotAverage3_nom (libgte), 8-102
 RotAverage4 (libgte), 8-103
 RotAverageNclip3 (libgte), 8-104
 RotAverageNclip3_nom (libgte), 8-105
 RotAverageNclip4 (libgte), 8-106
 RotAverageNclipColorCol3 (libgte), 8-107
 RotAverageNclipColorCol3_nom (libgte), 8-108
 RotAverageNclipColorDpq3 (libgte), 8-109
 RotAverageNclipColorDpq3_nom (libgte), 8-110
 RotColorDpq (libgte), 8-111
 RotColorDpq_nom (libgte), 8-112
 RotColorDpq3 (libgte), 8-113
 RotColorDpq3_nom (libgte), 8-114
 RotColorMatDpq (libgte), 8-115
 RotMatrix... (libgte), 8-116
 RotMatrix_gte (libgte), 8-118
 RotMatrixC (libgte), 8-119
 RotMatrixX (libgte), 8-120
 RotMatrixY (libgte), 8-121
 RotMatrixYXZ_gte (libgte), 8-122
 RotMatrixZ (libgte), 8-123
 RotMatrixZYX_gte (libgte), 8-124
 RotMeshH (libgte), 8-125
 RotMeshPrimQ_T (libgte), 8-126
 RotMeshPrimR_... (libgte), 8-127
 RotMeshPrimS_... (libgte), 8-128
 RotNclip3 (libgte), 8-129
 RotNclip3_nom (libgte), 8-130
 RotNclip4 (libgte), 8-131
 RotPMD_... (libgte), 8-132

RotPMD_SV_... (libgte), 8-133
 RotRMD_... (libgte), 8-134
 RotRMD_SV_... (libgte), 8-135
 RotSMD_... (libgte), 8-136
 RotSMD_SV_... (libgte), 8-137
 RotTrans (libgte), 8-138
 RotTrans_nom (libgte), 8-139
 RotTransPers (libgte), 8-140
 RotTransPers_nom (libgte), 8-141
 RotTransPers3 (libgte), 8-142
 RotTransPers3_nom (libgte), 8-143
 RotTransPers3N (libgte), 8-144
 RotTransPers4 (libgte), 8-145
 RotTransPers4_nom (libgte), 8-146
 RotTransPersN (libgte), 8-147
 RotTransSV (libgte), 8-148
 rsin (libgte), 8-149

S

ScaleMatrix (libgte), 8-150
 ScaleMatrixL (libgte), 8-151
 SelectGUN (libgun), 12-35
 SetBackColor (libgte), 8-152
 SetColorMatrix (libgte), 8-153
 SetConf (libapi), 1-62
 SetDefDispEnv (libgpu), 1-94
 SetDefDrawEnv (libgpu), 1-95
 SetDispMask (libgpu), 1-96
 SetDrawArea (libgpu), 1-97
 SetDrawEnv (libgpu), 1-98
 SetDrawLoad (libgpu), 1-99
 SetDrawMode (libgpu), 1-100
 SetDrawMove (libgpu), 1-101
 SetDrawOffset (libgpu), 1-102
 SetDrawStp (libgpu), 1-103
 SetDrawTPage (libgpu), 1-104
 SetDumpFnt (libgpu), 1-105
 SetFarColor (libgte), 8-154
 SetFogFar (libgte), 8-155
 SetFogNear (libgte), 8-156
 SetFogNearFar (libgte), 8-157
 SetGeomOffset (libgte), 8-158
 SetGeomScreen (libgte), 8-159
 SetGraphDebug (libgpu), 1-106
 setjmp (libc/libc2), 2-32
 SetLightMatrix (libgte), 8-160
 SetLineF2, SetLineF3, SetLineF4 (libgpu), 1-107
 SetLineG2, SetLineG3, SetLineG4 (libgpu), 1-107
 SetMem (libapi), 1-63
 SetMulMatrix (libgte), 8-161
 SetMulRotMatrix (libgte), 8-162
 SetPolyF3, SetPolyF4 (libgpu), 1-108
 SetPolyG3, SetPolyG4 (libgpu), 1-108
 SetPolyGT3, SetPolyGT4 (libgpu), 1-108
 SetRCnt (libapi), 1-64
 SetRGBcd (libgte), 8-163
 SetRotMatrix (libgte), 8-164
 SetSemiTrans (libgpu), 1-109
 SetShadeTex (libgpu), 1-110
 SetSp (libapi), 1-65
 SetSprt, SetSprt8, SetSprt16 (libgpu), 1-111
 SetTexWindow (libgpu), 1-112
 SetTile, SetTile1, SetTile8, SetTile16 (libgpu), 1-113
 SetTransMatrix (libgte), 8-165
 SetVideoMode (libetc), 12-36
 sin (libmath), 3-23
 sinh (libmath), 3-24
 Sio1Callback (libsio), 16-5
 sprintf (libc/libc2), 2-33
 sprintf2 (libmath), 3-25
 SpuClearReverbWorkArea (libspu), 15-15

SpuFlush (libspu), 15-16
 SpuFree (libspu), 15-17
 SpuGetAllKeysStatus (libspu), 15-18
 SpuGetCommonAttr (libspu), 15-19
 SpuGetCommonCDMix (libspu), 15-20
 SpuGetCommonCDReverb (libspu), 15-21
 SpuGetCommonCDVolume (libspu), 15-22
 SpuGetCommonMasterVolume (libspu), 15-23
 SpuGetCommonMasterVolumeAttr (libspu), 15-24
 SpuGetCommonMasterVolumeX (libspu), 15-25
 SpuGetIRQ (libspu), 15-26
 SpuGetIRQAddr (libspu), 15-27
 SpuGetKeyStatus (libspu), 15-28
 SpuGetMute (libspu), 15-29
 SpuGetNoiseClock (libspu), 15-30
 SpuGetNoiseVoice (libspu), 15-31
 SpuGetPitchLFOVoice (libspu), 15-32
 SpuGetReverb (libspu), 15-33
 SpuGetReverbModeDelayTime (libspu), 15-34
 SpuGetReverbModeDepth (libspu), 15-35
 SpuGetReverbModeFeedback (libspu), 15-36
 SpuGetReverbModeParam (libspu), 15-37
 SpuGetReverbModeType (libspu), 15-38
 SpuGetReverbVoice (libspu), 15-39
 SpuGetTransferMode (libspu), 15-40
 SpuGetTransferStartAddr (libspu), 15-41
 SpuGetVoiceADSR (libspu), 15-42
 SpuGetVoiceADSRAttr (libspu), 15-43
 SpuGetVoiceAR (libspu), 15-44
 SpuGetVoiceARAttr (libspu), 15-45
 SpuGetVoiceAttr (libspu), 15-46
 SpuGetVoiceDR (libspu), 15-47
 SpuGetVoiceEnvelope (libspu), 15-48
 SpuGetVoiceEnvelopeAttr (libspu), 15-49
 SpuGetVoiceLoopStartAddr (libspu), 15-50
 SpuGetVoiceNote (libspu), 15-51
 SpuGetVoicePitch (libspu), 15-52
 SpuGetVoiceRR (libspu), 15-53
 SpuGetVoiceRRAttr (libspu), 15-54
 SpuGetVoiceSampleNote (libspu), 15-55
 SpuGetVoiceSL (libspu), 15-56
 SpuGetVoiceSR (libspu), 15-57
 SpuGetVoiceSRAttr (libspu), 15-58
 SpuGetVoiceStartAddr (libspu), 15-59
 SpuGetVoiceVolume (libspu), 15-60
 SpuGetVoiceVolumeAttr (libspu), 15-61
 SpuGetVoiceVolumeX (libspu), 15-62
 Spulnit (libspu), 15-63
 SpulnitHot (libspu), 15-64
 SpulnitMalloc (libspu), 15-65
 SpulsReverbWorkAreaReserved (libspu), 15-66
 SpulsTransferCompleted (libspu), 15-67
 SpuLSetVoiceAttr (libspu), 15-68
 SpuMalloc (libspu), 15-69
 SpuMallocWithStartAddr (libspu), 15-70
 SpuNGetVoiceAttr (libspu), 15-71
 SpuNSetVoiceAttr (libspu), 15-72
 SpuQuit (libspu), 15-73
 SpuRead (libspu), 15-74
 SpuReadDecodedData (libspu), 15-75
 SpuReserveReverbWorkArea (libspu), 15-76
 SpuRGetAllKeysStatus (libspu), 15-77
 SpuRSetVoiceAttr (libspu), 15-78
 SpuSetCommonAttr (libspu), 15-79
 SpuSetCommonCDMix (libspu), 15-80
 SpuSetCommonCDReverb (libspu), 15-81
 SpuSetCommonCDVolume (libspu), 15-82
 SpuSetCommonMasterVolume (libspu), 15-83
 SpuSetCommonMasterVolumeAttr (libspu), 15-84
 SpuSetEnv (libspu), 15-85
 SpuSetESA (libspu), 15-86
 SpuSetIRQ (libspu), 15-87
 SpuSetIRQAddr (libspu), 15-88
 SpuSetIRQCallback (libspu), 15-89
 SpuSetKey (libspu), 15-90
 SpuSetKeyOnWithAttr (libspu), 15-91
 SpuSetMute (libspu), 15-92
 SpuSetNoiseClock (libspu), 15-93
 SpuSetNoiseVoice (libspu), 15-94
 SpuSetPitchLFOVoice (libspu), 15-95
 SpuSetReverb (libspu), 15-96
 SpuSetReverbDepth (libspu), 15-97
 SpuSetReverbModeDelayTime (libspu), 15-98
 SpuSetReverbModeDepth (libspu), 15-99
 SpuSetReverbModeFeedback (libspu), 15-100
 SpuSetReverbModeParam (libspu), 15-101
 SpuSetReverbModeType (libspu), 15-103
 SpuSetReverbVoice (libspu), 15-104
 SpuSetTransferCallback (libspu), 15-105
 SpuSetTransferMode (libspu), 15-106
 SpuSetTransferStartAddr (libspu), 15-107
 SpuSetVoiceADSR (libspu), 15-108
 SpuSetVoiceADSRAttr (libspu), 15-109
 SpuSetVoiceAR (libspu), 15-110
 SpuSetVoiceARAttr (libspu), 15-111
 SpuSetVoiceAttr (libspu), 15-112
 SpuSetVoiceDR (libspu), 15-115
 SpuSetVoiceLoopStartAddr (libspu), 15-116
 SpuSetVoiceNote (libspu), 15-117
 SpuSetVoicePitch (libspu), 15-118
 SpuSetVoiceRR (libspu), 15-119
 SpuSetVoiceRRAttr (libspu), 15-120
 SpuSetVoiceSampleNote (libspu), 15-121
 SpuSetVoiceSL (libspu), 15-122
 SpuSetVoiceSR (libspu), 15-123
 SpuSetVoiceSRAttr (libspu), 15-124
 SpuSetVoiceStartAddr (libspu), 15-125
 SpuSetVoiceVolume (libspu), 15-126
 SpuSetVoiceVolumeAttr (libspu), 15-127
 SpuStart (libspu), 15-128
 SpuStGetStatus (libspu), 15-129
 SpuStGetVoiceStatus (libspu), 15-130
 SpuStInit (libspu), 15-131
 SpuStQuit (libspu), 15-132
 SpuStSetPreparationFinishedCallback (libspu), 15-133
 SpuStSetStreamFinishedCallback (libspu), 15-134
 SpuStSetTransferFinishedCallback (libspu), 15-135
 SpuStTransfer (libspu), 15-136
 SpuWrite (libspu), 15-137
 SpuWrite0 (libspu), 15-138
 SpuWritePartly (libspu), 15-139
 sqrt (libmath), 3-26
 Square SL0 (libgte), 8-170
 Square SL12 (libgte), 8-171
 Square SS0 (libgte), 8-172
 Square SS12 (libgte), 8-173
 Square0 (libgte), 8-166
 Square12 (libgte), 8-167
 SquareRoot0 (libgte), 8-168
 SquareRoot12 (libgte), 8-169
 srand (libc/libc2), 2-34
 SsAllocateVoices (libsnd), 14-15
 SsBlockVoiceAllocation (libsnd), 14-16
 SsChannelMute (libsnd), 14-17
 SsEnd (libsnd), 14-18
 SsGetActualProgFromProg (libsnd), 14-19
 SsGetChannelMute (libsnd), 14-20
 SsGetCurrentPoint (libsnd), 14-21
 SsGetMute (libsnd), 14-22
 SsGetMVol (libsnd), 14-23
 SsGetNck (libsnd), 14-24
 SsGetRVol (libsnd), 14-25

SsGetSerialAttr (libsnd), 14-26
 SsGetSerialVol (libsnd), 14-27
 SsGetVoiceMask (libsnd), 14-28
 SslInit (libsnd), 14-29
 SslInitHot (libsnd), 14-30
 SslsEos (libsnd), 14-31
 SsPitchFromNote (libsnd), 14-32
 SsPlayBack (libsnd), 14-33
 SsQueueKeyOn (libsnd), 14-34
 SsQueueRegisters (libsnd), 14-35
 SsQueueReverb (libsnd), 14-36
 SsQuit (libsnd), 14-37
 SsSepClose (libsnd), 14-38
 SsSepOpen (libsnd), 14-39
 SsSepOpenJ (libsnd), 14-40
 SsSepPause (libsnd), 14-41
 SsSepPlay (libsnd), 14-42
 SsSepReplay (libsnd), 14-43
 SsSepSetAccelerando (libsnd), 14-44
 SsSepSetCrescendo (libsnd), 14-45
 SsSepSetDecrescendo (libsnd), 14-46
 SsSepSetRitardando (libsnd), 14-47
 SsSepSetVol (libsnd), 14-48
 SsSepStop (libsnd), 14-49
 SsSeqCalledTbyT (libsnd), 14-50
 SsSeqClose (libsnd), 14-51
 SsSeqGetVol (libsnd), 14-52
 SsSeqOpen (libsnd), 14-53
 SsSeqOpenJ (libsnd), 14-54
 SsSeqPause (libsnd), 14-55
 SsSeqPlay (libsnd), 14-56
 SsSeqPlayPtoP, 14-57
 SsSeqReplay (libsnd), 14-58
 SsSeqSetAccelerando (libsnd), 14-59
 SsSeqSetCrescendo (libsnd), 14-60
 SsSeqSetDecrescendo (libsnd), 14-61
 SsSeqSetNext (libsnd), 14-62
 SsSeqSetRitardando (libsnd), 14-63
 SsSeqSetVol (libsnd), 14-64
 SsSeqSkip, 14-65
 SsSeqStop (libsnd), 14-66
 SsSetAutoKeyOffMode (libsnd), 14-67
 SsSetCurrentPoint (libsnd), 14-68
 SsSetLoop (libsnd), 14-69
 SsSetMarkCallback (libsnd), 14-70
 SsSetMono (libsnd), 14-71
 SsSetMute (libsnd), 14-72
 SsSetMVol (libsnd), 14-73
 SsSetNck (libsnd), 14-74
 SsSetNext (libsnd), 14-74
 SsSetNoiseOff (libsnd), 14-24
 SsSetNoiseOn (libsnd), 14-24
 SsSetReservedVoice (libsnd), 14-75
 SsSetRVol (libsnd), 14-76
 SsSetSerialAttr (libsnd), 14-77
 SsSetSerialVol (libsnd), 14-78
 SsSetStereo (libsnd), 14-79
 SsSetTableSize (libsnd), 14-80
 SsSetTempo (libsnd), 14-81
 SsSetTickCallBack (libsnd), 14-82
 SsSetTickMode (libsnd), 14-83
 SsSetVoiceMask (libsnd), 14-84
 SsSetVoiceSettings (libsnd), 14-85
 SsStart (libsnd), 14-86
 SsStart2 (libsnd), 14-87
 SsUnBlockVoiceAllocation (libsnd), 14-88
 SsUtAllKeyOff (libsnd), 14-89
 SsUtAutoPan (libsnd), 14-90
 SsUtAutoVol (libsnd), 14-91
 SsUtChangeADSR (libsnd), 14-92
 SsUtChangePitch (libsnd), 14-93
 SsUtFlush (libsnd), 14-94
 SsUtGetDetVVol (libsnd), 14-95
 SsUtGetProgAtr (libsnd), 14-96
 SsUtGetReverbType (libsnd), 14-97
 SsUtGetVabHdr (libsnd), 14-98
 SsUtGetVagAddr (libsnd), 14-99
 SsUtGetVagAddrFromTone (libsnd), 14-100
 SsUtGetVagAtr (libsnd), 14-101
 SsUtGetVBaddrInSB (libsnd), 14-102
 SsUtGetVVol (libsnd), 14-103
 SsUtKeyOff (libsnd), 14-104
 SsUtKeyOffV (libsnd), 14-105
 SsUtKeyOn (libsnd), 14-106
 SsUtKeyOnV (libsnd), 14-107
 SsUtPitchBend (libsnd), 14-108
 SsUtReverbOff (libsnd), 14-109
 SsUtReverbOn (libsnd), 14-110
 SsUtSetDetVVol (libsnd), 14-111
 SsUtSetProgAtr (libsnd), 14-112
 SsUtSetReverbDelay (libsnd), 14-113
 SsUtSetReverbDepth (libsnd), 14-114
 SsUtSetReverbFeedback (libsnd), 14-115
 SsUtSetReverbType (libsnd), 14-116
 SsUtSetVabHdr (libsnd), 14-117
 SsUtSetVagAtr (libsnd), 14-118
 SsUtSetVVol (libsnd), 14-119
 SsVabClose (libsnd), 14-120
 SsVabFakeBody (libsnd), 14-121
 SsVabFakeHead (libsnd), 14-122
 SsVabOpen (libsnd), 14-123
 SsVabOpenHead (libsnd), 14-124
 SsVabOpenHeadSticky (libsnd), 14-125
 SsVabTransBody (libsnd), 14-126
 SsVabTransBodyPartly (libsnd), 14-127
 SsVabTransCompleted (libsnd), 14-128
 SsVabTransfer (libsnd), 14-129
 SsVoiceCheck (libsnd), 14-130
 SsVoKeyOff (libsnd), 14-131
 SsVoKeyOn (libsnd), 14-132
 StartCARD (libcard), 4-4
 StartGUN (libgun), 12-37
 StartPAD (libapi), 1-66
 StartRCnt (libapi), 1-67
 StartTAP (libtap), 12-38
 StCdInterrupt (libcd), 10-45
 StClearRing (libcd), 10-46
 StFreeRing (libcd), 10-47
 StGetBackloc (libcd), 10-48
 StGetNext (libcd), 10-49
 StGetNextS (libcd), 10-50
 StNextStatus (libcd), 10-51
 StopCallback (libetc), 12-39
 StopCARD (libcard), 4-5
 StopGUN (libgun), 12-40
 StopPAD (libapi), 1-68
 StopRCnt (libapi), 1-69
 StopTAP (libtap), 12-41
 StoreImage (libgpu), 1-114
 StoreImage2 (libgpu), 1-115
 strcat (libc/libc2), 2-35
 strchr (libc/libc2), 2-36
 strcmp (libc/libc2), 2-37
 strcpy (libc/libc2), 2-38
 strcspn (libc/libc2), 2-39
 StRingStatus (libcd), 10-52
 strlen (libc/libc2), 2-40
 strncat (libc/libc2), 2-41
 strncmp (libc/libc2), 2-42
 strncpy (libc/libc2), 2-43
 strpbrk (libc/libc2), 2-44
 strchr (libc/libc2), 2-45

strspn (libc/libc2), 2-46
 strstr (libc/libc2), 2-47
 strtod (libc/libc2), 3-27
 strtok (libc/libc2), 2-48
 strtol (libc/libc2), 2-49
 strtoul (libc/libc2), 2-50
 StSetChannel (libcd), 10-53
 StSetEmulate (libcd), 10-54
 StSetMask (libcd), 10-55
 StSetRing (libcd), 10-56
 StSetStream (libcd), 10-57
 StUnSetRing (libcd), 10-58
 SubPol3 (libgte), 8-174
 SubPol4 (libgte), 8-175
 SwEnterCriticalSection (libapi), 1-70
 SwExitCriticalSection (libapi), 1-71
 SystemError (libapi), 1-72

T

tan (libmath), 3-28
 tanh (libmath), 3-29
 TermPrim (libgpu), 1-116
 TestEvent (libapi), 1-73

Macros

A

addPrim (libgpu), 1-33
 addPrims (libgpu), 1-34
 addVector (libgpu), 1-119
 applyVector (libgpu), 1-120

C

catPrim (libgpu), 1-36
 CombAsyncRequest (libcomb), 13-8
 CombBytesRemaining (libcomb), 13-9
 CombBytesToRead (libcomb), 13-10
 CombBytesToWrite (libcomb), 13-11
 CombCancelRead (libcomb), 13-12
 CombCancelWrite (libcomb), 13-13
 CombControlStatus (libcomb), 13-14
 CombCTS (libcomb), 13-15
 CombGetBPS (libcomb), 13-16
 CombGetMode (libcomb), 13-17
 CombGetPacketSize (libcomb), 13-18
 CombReset (libcomb), 13-19
 CombResetError (libcomb), 13-20
 CombResetVBLANK (libcomb), 13-21
 CombSetBPS (libcomb), 13-22
 CombSetControl (libcomb), 13-23
 CombSetMode (libcomb), 13-24
 CombSetPacketSize (libcomb), 13-25
 CombSetRTS (libcomb), 13-26
 CombSioStatus (libcomb), 13-27
 CombWaitCallback (libcomb), 13-28
 copyVector (libgpu), 1-121

D

dump... (libgpu), 1-125
 dumpClut (libgpu), 1-50
 dumpMatrix (libgpu), 1-122
 dumpRECT (libgpu), 1-123
 dumpTPage (libgpu), 1-54
 dumpVector (libgpu), 1-124

TransMatrix (libgte), 8-176
 TransposeMatrix (libgte), 8-177
 TransRot_32 (libgte), 8-180
 TransRotPers (libgte), 8-178
 TransRotPers3 (libgte), 8-179

U

undelete (libapi), 1-74
 UndeliverEvent (libapi), 1-75

V

VectorNormal (libgte), 8-181
 VectorNormalS (libgte), 8-182
 VectorNormalSS (libgte), 8-183
 VSync (libetc), 1-117
 VSyncCallback (libetc), 1-118

W

WaitEvent (libapi), 1-76
 write (libapi), 1-77

F

fabs (libmath), 3-12

G

getClut (libgpu), 1-59
 getTPage (libgpu), 1-70
 GsClearDispArea (libgs), 1-109
 GsIncFrame (libgs), 1-110
 GsSetAzwh (libgs), 1-111

I

isendprim (libgpu), 1-71
 isXXXX... (libc/libc2), 2-16

N

nextPrim (libgpu), 1-86

S

setClut (libgpu), 1-126
 setDrawTPage (libgpu), 1-104
 setLineF2, setLineF3, setLineF4 (libgpu), 1-107
 setLineG2, setLineG3, setLineG4 (libgpu), 1-107
 setPolyF3, setPolyF4 (libgpu), 1-108
 setPolyG3, setPolyG4 (libgpu), 1-108
 setPolyGT3, setPolyGT4 (libgpu), 1-108
 setRECT (libgpu), 1-127
 setRGB0, setRGB1, setRGB2, setRGB3 (libgpu), 1-128
 setSemiTrans (libgpu), 1-109
 setShadeTex (libgpu), 1-110
 setSprt, setSprt8, setSprt16 (libgpu), 1-111
 setTexWindow (libgpu), 1-112
 setTile, setTile1, setTile8, setTile16 (libgpu), 1-113
 setTPage (libgpu), 1-129
 setUV0, setUV3, setUV4 (libgpu), 1-130
 setUVWH (libgpu), 1-131
 setVector (libgpu), 1-132
 setWH (libgpu), 1-133
 setXY0, setXY2, setXY3, setXY4 (libgpu), 1-134
 setXYWH (libgpu), 1-135

10 XXX Library Functions

T

termPrim (libgpu), 1-116

toascii (libc/libc2), 2-51
tolower (libc/libc2), 2-52
toupper (libc/libc2), 2-53